Răzvan Rughiniş
Răzvan Deaconescu

George Milescu
Mircea Bardac

Computer Science

Utilizarea Sistemelor de Operare

Editura PRINTECH

Material cu drept de autor

Cuvânt înainte

Sistemul de operare reprezintă, probabil, unul din termenii cei mai des întâlniți în domeniul calculatoarelor și nu numai. Nu poate surprinde faptul că sistemele de operare sunt un bun exemplu al diversității de care s-a bucurat lumea informaticii. Sistemele de operare au cunoscut o evoluție intensă de-a lungul ultimelor decenii. Dacă în anii '60 sistemele de operare reprezentau un simplu termen în jargonul profesioniștilor în calculatoare, cu o interfață greoaie și neprietenoasă, în ziua de astăzi sistemele de operare au o interfață extrem de facilă pe o gamă din ce în ce mai diversă de dispozitive (calculatoare, PDA-uri, telefoane mobile) și pentru diverse cerințe (programare, media, gaming, servere).

Cartea de față își propune un prim contact cu lumea sistemelor de operare și, în mod particular, cu latura preponderent tehnică a acestora. Am creat această carte având în permanență în vedere cunoștințele de bază și cadrul conceptual care sunt utile unui student la o facultate de calculatoare. Dincolo de conținut, sperăm că această carte să ofere un set de deprinderi și de abordări în soluționarea de probleme care depășesc sfera sistemelor de operare și a utilizării acestora.

Diversitatea subiectelor abordate în această carte a făcut foarte dificilă crearea unei succesiuni clare de capitole. Abordarea aleasă este una stratificată, fiecare capitol bazându-se pe capitolele studiate anterior. Totuși, au existat momente în care anumite noțiuni au trebuit utilizate înainte de a fi definit cadrul conceptual sau câteva capitole după ce au fost prezentate. Pentru aceasta, cititorului îi sunt oferite referințe către alte capitole pentru a clarifica anumite noțiuni sau concepte proaspăt prezentate. Mai mult, dorim ca această carte să ofere o colecție de materiale utile dincolo de cadrul oferit de cursul de Utilizarea Sistemelor de Operare. Nu impunem ca toate informațiile prezente aici să fie asimilate pe parcursul cursului de Utilizarea Sistemelor de Operare, dar sperăm ca studentul dornic de aprofundare să găsească un bun prilej de răsfoire a acestei cărți și după absolvirea cursului.

Din punct de vedere tehnic, materialul de față oferă o perspectivă ce aparține preponderent universului Linux. Am considerat contactul cu Linux ca fiind o oportunitate aparte pentru o mare majoritate a utilizatorilor ce provin din mediul Windows, pentru care alternativele în domeniul sistemelor de operare nu au reprezentat până acum o opțiune luată în considerație. Dorința noastră este ca utilizarea unui nou sistem de operare cu o răspândire și evoluție din ce în ce mai intensă să ofere o nouă perspectivă asupra lumii calculatoarelor în general și a sistemelor de operare în particular. Deși se insistă pe Linux, fiecare capitol are secțiuni de studii de caz în care sunt prezentate mecanismele similare pe un sistem Windows.

Stuctura cărții își propune să ofere atât o prezentare a cadrului conceptual, cât și parte aplicativă bogată în exemple. Am considerat că este benefic pentru un material de suport pentru cursul de Utilizarea Sistemelor de Operare să se prezinte ca un set bine definit de exemple de utilizare a diverselor comenzi sau aplicații. Fiecare capitol este prefațat cu o mică secțiune "Ce se învață în acest capitol?" utilă pentru prezentarea principalelor idei. De asemenea, orice capitol se încheie cu o secțiune de "Cuvinte cheie" și una de "Întrebări" cu rol de autoevaluare a cititorului.

Mulțumim tuturor celor care și-au adus contribuția la conținutul cărții. Această primă editie se bucură acum de experiența și pasiunea unei echipe entuziaste. În primul rând, îi mulțumim domnului profesor Nicolae Țăpuș, precum și colegilor noștri: Ștefan Bucur, Andrei Ciorba, Bogdan Doinea, Alex Mihăiuc. Totodată, mulțumim studenților ce au ales drept început al pregătirii lor această facultate. Ei sunt cei cărora le dedicăm această carte. Ne-a făcut foarte mare plăcere să elaborăm această carte și sperăm ca măcar o parte din entuziasmul nostru să îl regăsiți în paginile cărții.

Autorii

1 Introducere

It's important in the long run to have multiple languages, multiple operating systems, and if you don't have that competition, then we don't survive.

Larry Wall

1.1 Ce este un sistem de operare?

Sistemele de operare reprezintă un concept familiar tuturor utilizatorilor de calculatoare personale. Ele apar tot mai des şi în implementarea de funcţionalităţi din alte aparate, de la telefoanele mobile la cardurile bancare inteligente. Există însă şi dispozitive care nu au (încă) un sistem de operare, în principal deoarece execută un număr mic de operaţiuni care nu necesită nici o modificare. De exemplu, calculatorul dintr-un cuptor cu microunde execută un program simplu fixat prin circuite fizice. Sistemele de operare sunt utile în cazul calculatoarelor care trebuie să se adapteze unui mediu în schimbare, ele permiţând modificarea comportamentului calculatorului fără schimbarea succesiunii operaţiilor hardware.

Un sistem de operare *este un program care controlează distribuţia resurselor unui calculator şi mediază comunicarea dintre hardware, pe de o parte, şi aplicaţiile utilizatorilor, pe de altă parte.*

Scopul unui SO este să asigure stabilitate şi flexibilitate în funcţionarea calculatorului. Sistemele de operare permit, în general, interacţiunea cu mai mulţi utilizatori, rularea mai multor aplicaţii, şi modificarea strategiilor de răspuns ale calculatorului la o anumită problemă.

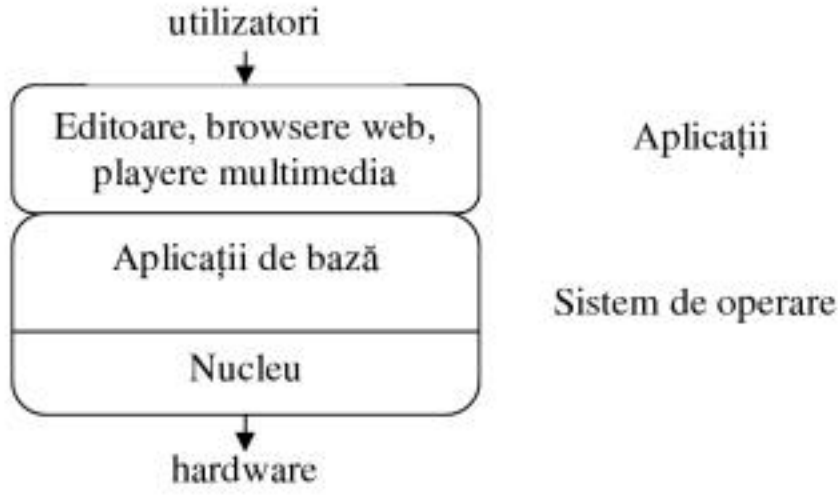


Figura 1-1 Rolul unui sistem de operare

În figura de mai sus este prezentat rol de mediere al sistemului de operare între utilizatori şi hardware. Sistemul de operare va avea acces direct la resursele hardware, în vreme ce

utilizatorii folosesc cel mai adesea aplicații ce apelează prin intermediul bibliotecilor sau direct funcții ale sistemului de operare.

Componenta esențială a sistemului de operare o reprezintă **nucleul** sau kernelul. Acesta are rolul de gestiune a resurselor fizice şi de oferire a unei interfețe comune aplicațiilor. Nucleul nu este o aplicație în sine, ci este un mediu în cadrul căruia pot rula alte aplicații.

Programele de bază sunt aplicațiile fundamentale ale unui sistem de operare care permit interacțiunea cu nucleul şi cu sistemul fizic. Astfel de aplicații sunt interpretoarele de comenzi, utilitare de gestiune a utilizatorilor şi a sistemului de fişiere, biblioteci, etc. În general nu este bine conturată definiția de aplicație de bază; se consideră că o aplicație este o aplicație de bază atunci când oferă servicii pentru o bună parte din celelalte componente ale sistemului.

Din punct de vedere tehnic sistemul de operare este chiar nucleul. Din perspectiva utilizatorului sitemul de operare este compus din nucleu şi suita programelor de bază. Programele de baza sunt acele aplicații ce permit utilizatorului interacțiunea cu resursele hardware ale sistemului, această relație fiind ilustrată în figura de mai sus.

Prin analogie un calculator poate fi privit ca un aeroport cu un punct central din care se coordonează traficul. Resursele aeroportului le reprezintă pistele şi avioanele iar buna funcționare a aeroportului constă în desfăşurarea fluentă şi eficientă a traficului aerian. Fără un control organizat al acestor resurse, pe un aeroport nu s-ar putea evita întârzierile şi coliziunile. În cazul calculatoarelor, rolul unui sistem de operare este de a coordona accesul la resurse şi traficul, întocmai precum turnul de control al unui aeroport.

1.1.1 Funcțiile unui sistem de operare

O primă funcție a unui sistem de operare constă în **administrarea resurselor hardware** ale sistemului.

În cazul unui calculator, este vorba mai ales despre accesul la procesor, memorie, hard-disk, comunicarea în rețea, precum şi cu alte dispozitivele de intrare/ieşire. Aceste resurse pot fi solicitate de aplicații multiple şi de utilizatori multipli.

Sistemul de operare este el însuşi o aplicație, care necesită resurse (şi, implicit, accesul la ele) pentru a funcționa, ca şi celelalte aplicații; prin urmare, acesta trebuie să îşi asigure accesul privilegiat la resurse. Un SO trebuie să mențină controlul asupra calculatorului şi să facă posibilă utilizarea sa predictibilă şi cât mai eficientă.

Sistemul de operare oferă o **abstractizare a resurselor hardware**. Practic SO este o extensie a sistemului fizic, oferind o inferfață simplificată folosită pentru medierea accesului utilizatorilor şi aplicațiilor la resursele hardware.

Sistemul de operare este responsabil cu oferirea unui set limitat de informații şi funcții de lucru cu hardware-ul, în loc de descrierea exactă a ceea ce se întâmplă în hardware. Practic, el oferă dispozitivelor şi aplicațiilor (şi, în ultimă instanță, utilizatorilor) o transparență față de modul fizic de funcționare a hardware-ului. În ceea ce priveşte comunicarea dintre hardware şi aplicații, sistemul de operare trebuie să creeze o identitate calculatorului, care să rămână relativ stabilă în dialogul acestuia cu aplicații diverse, chiar dacă au loc upgrade-uri de hardware. Sistemul de operare este şi cel care permite unei aplicații să ruleze pe sisteme fizice diferite, deoarece uniformizează răspunsurile date de calculator şi intermediază accesul la resurse. Sistemele de operare trebuie prin urmare să poată interacționa cu arhitecturi hardware şi cu periferice foarte diverse, el fiind cel care facilitează accesul aplicațiilor la resursele hardware. Această funcție se numeşte **portabilitate** şi se referă la oferirea unei interfețe unice pentru utilizator, indiferent de diferențele hardware dintre arhitecturi.

Un SO foloseşte *drivere* pentru **comunicarea cu dispozitive periferice**. Acestea sunt programe care traduc comenzile sistemului de operare în comenzi inteligibile echipamentelor, precum şi ieşirea acestora în mesaje accesibile sistemului.

Rolul SO a fost extins prin **punerea la dispoziția aplicațiilor a unei colecții de biblioteci** (funcții) pe care programele le pot folosi pentru a rezolva probleme specifice legate de interacțiunea cu componentele calculatorului.

Sistemul de operare va include **utilitare de diagnosticare şi monitorizare** a funcționării diferitelor componente atât pentru cele hardware, cât şi pentru componentele software.

Un sistem de operare poate, desigur, să şi eşueze în realizarea funcțiilor sale. Un tip foarte frecvent de eroare constă în blocarea funcționării calculatorului. Un astfel de „crash" poate fi datorat unor imperfecțiuni ale sistemului, dar poate fi favorizat şi de funcționarea anumitor aplicații. Un al doilea tip frecvent de eroare se referă la problemele de securitate, care pot permite intruziunea unui utilizator sau a unei aplicații malițioase. De asemenea, este posibil ca un sistem de operare să nu reuşească să interacționeze corect cu anumite periferice sau echipamente.

În principal, **funcționarea corectă a unui sistem de operare** poate fi păstrată prin realizarea metodică a unui număr limitat de operații:

- repararea problemelor (bug-urilor) din sistemul de operare însuşi;
- repararea bug-urilor din aplicații;
- eliminarea viruşilor, a viermilor, a Spyware etc;
- instalarea de versiuni actualizate ale sistemul de operare şi ale aplicațiilor acestuia.

1.1.2 Tipuri de sisteme de operare

O clasificare frecventă a sistemelor de operare include şase categorii, în funcție de nevoile principale adresate.

Sistemele de operare de pe smart carduri sunt cele mai simple, servind deseori funcții reduse (precum în cazul cărților de debit) şi administrând resurse foarte limitate. Totuşi, ele oferă o un management de bază al acestor resurse, inclusiv a spațiului de stocare, putând oferi şi servicii de securitate/criptare a datelor ce le dețin.

Sistemele de operare embedded sunt incluse în dispozitive precum televizoarele sau telefoanele mobile, având constrângeri specifice funcționalității lor. Sistemele de operare pentru dispozitivele embedded variază de la sisteme dedicate unui singur scop, ce nu oferă o interfață de interacțiune utilizatorului, până la sisteme multitasking (precum în cazul sistemelor de operare pentru PDA) ce oferă multe dintre fucționalitățile sistemelor de operare pentru calculatoarele personale.

Sistemele de operare de timp real (RTOS – Real Time Operating Systems) sunt folosite mai ales în echipamente ştiințifice, industriale şi de precizie. Scopul principal al acestora este să optimizeze utilizarea resurselor echipamentelor şi să asigure predictibilitatea funcționării lor, dat fiind că eşecurile în coordonarea subcomponentelor pot avea consecințe dezastruoase. Timpul este o variabilă esențială, deoarece anumite sarcini trebuie realizate într-un interval precis pentru a se putea coordona cu alte sarcini. Astfel de sisteme de operare nu sunt în general menite să permită comunicarea cu utilizatorul, dat fiind că nu este necesar să fie modificate, ele fiind concepute strict pentru executarea anumitor procese tehnologice. RTOS pot fi împărțite în sisteme de timp real hard, prevalente în procesele industriale, în care constrângerile temporale sunt întotdeauna prioritare, şi sisteme de timp real soft, precum sistemele dedicate multimedia, în care discordanțe temporale minore sunt acceptabile sau insesizabile.

Există numeroase sisteme de operare de timp real, însă proiectele ce captează încă atenția unui număr impresionant de dezvoltatori sunt TRON şi QNX. TRON a fost multă vreme dezvoltat în mediul academic din Japonia, pentru a îşi găsi mai apoi importante aplicații în industrie. Surselele de kernel pentru QNX au devenit din septembrie 2007 publice.

Sistemele de operare pentru un singur utilizator şi un singur task (single-user, single task) îşi propun să permită utilizarea optimă a resurselor calculatorului pentru un singur utilizator care rulează o singură aplicație la un moment dat. Aceste sisteme de operare în

prezent au doar relevanță didactică datorită simplității de interacțiune între resursele hardware și operațiile solicitate de către utilizator.

Sistemele de operare pentru mai mulți utilizatori (multi-user) urmăresc în principal să optimizeze funcționarea calculatorului prin echilibrarea cerințelor utilizatorilor și izolarea problemelor acestora. Acestea sunt diferite de sistemele de operare cu un singur utilizator dar care permit lucrul în rețea, deoarece în acest caz din punctul de vedere al sistemului de operare nu există decât un singur utilizator, și anume administratorul de sistem. Există sisteme de operare specializate pentru situațiile de conectare a calculatoarelor în rețea, precum și sisteme de operare specializate pentru aplicațiile care rulează pe un server, deși aceste granițe tind să se estompeze. [vezi 4.1.4]

În această categorie sunt incluse sistemele de operare precum cele din familiile Microsoft Windows, Unix și Apple Macintosh, ce au ca scop optimizarea folosirii resurselor atunci când un utilizator rulează concomitent mai multe aplicații sau când mai mulți utilizatori interacționează cu un calculator în același timp. Acestea sunt cele mai familiare pentru utilizatorii de PC-uri.

O practică uzuală pentru obținerea unei puteri sporite de procesare o reprezintă instalarea de procesoare multiple într-un singur sistem. Pentru managementul eficient al acestora și pentru a se putea folosi întreaga putere de procesare disponibilă, există **sisteme de operare pentru arhitecturi multi-procesor** special concepute pentru procesarea paralelă (simultană). Acestea pot fi variante modificate ale sistemelor de operare pentru servere, optimizate pentru gestiunea paralelă a operațiilor pe care le execută. Multe dintre sistemele de operare conteporane pot fi folosite atât pe arhitectruri monoprocesor, cât și pe cele multi-procesor, spre exemplu Microsft Vista sau distribuții bazate pe Linux 2.6.x.

1.1.3 Funcționarea sistemelor de operare

Un sistem de operare realizează următoarele sarcini principale: administrarea procesorului, administrarea memoriei, administrarea echipamentelor și a perifericelor, administrarea sistemelor de stocare a datelor, interfața cu aplicațiile și interfața cu utilizatorii. Scopul principal constă în impunerea de reguli pentru folosirea în comun a resurselor (denumită și multiplexare). Aceasta se realizează fie prin alternarea în timp a accesului la resurse (multiplexare în timp), de exemplu în cazul administrării procesorului, fie prin delimitarea și alocarea unor segmente din resurse aplicațiilor ce le vor utiliza simultan (multiplexare în spațiu, în cazul memoriei).

De asemenea, sistemele de operare conțin utilitare standard care permit realizarea unor sarcini precum:

- inițializarea sau oprirea proceselor, precum și comunicarea între procese;
- organizarea aplicațiilor disponibile pe calculator;
- organizarea fișierelor în directoare;
- vizualizarea și editarea fișierelor, precum și redenumirea, copierea sau ștergerea fișierelor.

Administrarea procesorului vizează optimizarea accesului diferitelor unități de software la ceasul procesorului. Este important de observat aici că, deși sistemele de operare multi-tasking permit funcționarea simultană a mai multor aplicații, aceasta este de fapt simulată prin comutarea foarte rapidă de la o aplicație la alta a unui procesor single-tasking. Execuția simultană în adevăratul sens al cuvântului are loc numai pe arhitecturi multiprocesor, unde fiecare procesor la un moment dat executa un proces. Prin urmare, este esențial ca sistemul de operare să fie responsiv la solicitările diferitelor aplicații. De asemenea, sistemul de operare trebuie să minimizeze proporția de timp în care procesorul este utilizat pentru execuția sarcinilor de întreținere, de comunicare internă, etc, în favoarea timpului alocat cerințelor directe ale utilizatorului.

Unitățile de software cu care lucrează un sistem de operare nu sunt aplicațiile, ci procesele sau firele de execuție (threads). Un proces este un program care execută o acțiune şi este controlat de utilizator, de o aplicație sau de sistemul de operare. O aplicație poate fi ea însăşi un proces dar poate fi la rândul său inițiatoarea altor procese pentru a comunica cu alte calculatoare sau cu alte aplicații. Sistemele de operare inițiază ele însele numeroase procese pentru a administra resursele calculatorului, procese de care utilizatorii pot să nu fie conştienți.

Administrarea memoriei urmăreşte să asigure faptul că fiecare proces are suficientă memorie, fără să invadeze spațiul de memorie al altor procese. De asemenea, procesele trebuie să folosească adecvat diferitele tipuri de memorie existentă – ținând cont de accesibilitatea diferențiată a acestora.

Administrarea perifericelor şi a echipamentelor este realizată prin intermediul driverelor. Acestea sunt programe relativ autonome de sistemul de operare, fiind deseori furnizate de producătorul echipamentelor şi instalate separat. Ele permit sistemului de operare să comunice cu echipamente noi fără să fie modificat. Comunicarea cu mediul este intermediată de spații tampon (buffere), în care datele de intrare sunt acumulate urmând să fie transmise procesorului în ritmul accesibil acestuia, evitând congestia atunci când apare o aglomerare datorită multitudinii de aplicații.

Funcția de mediere a sistemului de operare are două subcomponente:

- interfața cu aplicațiile (API – Application Programming Interface) permite aplicațiilor să acceseze facil resursele hardware ale sistemului fără a controla toate detaliile tehnice ale aplicațiilor;
- interfața cu utilizatorul permite acestuia din urmă să comunice cu calculatorul, fie prin comenzi text (TUI – Text User Interface) sau printr-o interfață grafică (GUI – Graphical User Interface).

1.2 Scurt istoric al sistemelor de operare

Sistemele de operare pentru PC-urile actuale sunt nu numai rezultatul unei evoluții graduale, ci şi al unei schimbări conceptuale, în măsura în care sistemul propriu-zis, şi anume kernel-ul, este acompaniat de interfața grafică şi de aplicații diverse cu scopul de a fi cât mai accesibil utilizatorului individual neprofesionist. Sistemele de timp real şi cele dedicate (embedded) sunt mai apropiate de formele inițiale, dezvoltate în anii 50-60.

1.2.1 Prima generație

Pot fi distinse patru mari perioade în evoluția sistemelor de operare (Tanenbaum 2001). În prima perioadă, care s-a desfăşurat în mare între anii 1945 şi 1955, se dezvoltă prima generație de calculatoare, având la bază **tehnologia tuburilor vidate**. O inovație majoră este introducerea cartelelor perforate, ca modalitate de stocare a informației. În această etapă nu există nici un sistem de operare şi nici măcar limbaje de programare; utilizatorul final este responsabil pentru interacțiunea directă cu hardware-ul.

1.2.2 A doua generație

Între anii 1955 şi 1965 introducerea **tranzistorului** marchează trecerea la a doua generație de calculatoare, dominate de sistemele mainframe. Principalele limbaje utilizate sunt FORTRAN şi limbajul de ansamblare. Sistemele de operare precum FMS (Fortran Monitor System) sau IBSYS oferă mai ales asistență pentru operațiile de input şi output. În această perioadă are loc o primă extindere semnificativă a utilizării calculatoarelor, soldată cu apariția diviziunii muncii între designeri, constructori, operatori, programatori şi personalul de

întreținere. Unitatea operațională principală de structurare a informației în interacțiunea cu calculatorul este job-ul, care consta într-unul sau mai multe programe scrise întâi pe hârtie şi apoi introduse pe cartele. O proporție substanțială din timpul de utilizare al mainframe-ului se pierde cu introducerea fizică a joburilor în calculator (prin realizarea şi mutarea cartelelor). Pentru a reduce acest incovenient se dezvoltă tehnica procesării agregate a job-urilor (în batch-uri): mai multe job-uri sunt grupate într-un singur şir de cartele separate de delimitatori. Job-urile sunt ulterior imprimate pe bandă magnetică folosind un calculator relativ ieftin, apoi benzile sunt transmise mainframe-ului, urmând ca outputul să fie tipărit din nou prin intermediul calculatorului mai ieftin. Primul sistem de operare major bazat pe principiul „*single stream batch processing*" a fost dezvoltat de General Motors Research Laboratories pentru mainframe-ul IBM 701 în 1956.

1.2.3 A treia generație

A treia generație de calculatoare, bazată pe **circuitele integrate**, apare în perioada 1965-1980. Principiul „multiprogramming" eficientizează folosirea resurselor calculatorului de către mai multe job-uri: când un job ocupă timpul de procesor, celelalte pot realiza operații de intrare/ieşire, şi vice-versa. Sistemul de operare OS/360 al IBM a realizat acest lucru prin partiționarea memoriei: mai multe programe puteau fi astfel introduse simultan în memorie. O altă optimizare este introdusă de tehnica spooling (stocarea job-urilor sub forma unei cozi, într-o memorie temporară), care permitea citirea directă a cartelelor, fără intermedierea unui alt calculator.

Principiul utilizării în comun a timpului de procesor (timesharing) este o altă inovație a acestei perioade, în care mai mulți utilizatori folosesc simultan terminale conectate la acelaşi calculator. Deşi scopul este acelaşi, şi anume optimizarea folosirii resurselor, tehnica utilizată diferă față de multiprogramming. În multiprogramming fiecare job este executat până se ajunge la operațiuni de input/output, moment în care procesorul e utilizat de alt job. În timesharing fiecare utilizator primeşte alternativ o fracțiune de timp, perioadă în care procesorul execută instrucțiunile acestuia. Astfel se valorifică eficient timpii morți datorați vitezei scăzute de acțiune a oamenilor față de cea a procesorului, reducând substanțial durata de elaborare şi corectare a unui program. Primul sistem bazat pe timesharing a fost CTSS – Compatible Time-Sharing System, realizat de MIT în 1961. Din sistemul de operare Multics, dezvoltat ulterior tot de MIT, au derivat primele variante de UNIX, inițial sub conducerea lui Ken Thompson. Dennis Ritchie elaborează limbajul C special pentru redactarea UNIX, evitând astfel dependența de caracteristicile hardware, inerentă în folosirea limajului de ansamblare. UNIX a fost primul sistem de operare scris într-un limbaj de nivel superior, devenind astfel portabil de la un tip de computer la altul. Datorită nevoii de standardizare a multiplelor variante de sisteme UNIX care au apărut apoi independent pe baza codului sursă, IEEE a elaborat standardul POSIX. Ulterior Linus Torvalds a scris Linux.

Tot în această perioadă, eşecurile notabile în elaborarea unor sisteme de operare foarte complexe, precum OS/360 al IBM, aduc noi schimbări şi perspective în ceea ce priveşte ingineria software. La nivel teoretic sunt cristalizate trei paradigme în designul sistemelor de operare. Modelul designului structural porneşte de la nevoia de a concepe sistemul ca un întreg şi de a-l planifica în ansamblul său; pe de altă parte, modelul dezvoltării graduale accentuează nevoia de feed-back şi adaptare la nevoi pe măsură ce acestea devin manifeste. Modelul metodelor formale subliniază utilitatea instrumentelor matematice de a simplifica procesul de design al sistemelor de operare.

În 1968 a fost creată firma Intel, specializată în microprocesoare. CP/M (Control Program for Microcomputers), elaborat în 1976 de Gary Kildall, este primul sistem de operare pentru calculatoare bazate pe procesoare Intel.

1.2.4 A patra generație

După 1980, până în prezent, se poate vorbi despre a patra generație de calculatoare, reprezentate de apariția și expansiunea extraordinară a **calculatoarelor personale**. Producția acestora are la bază tehnologia microprocesoarelor.

Bill Gates a cumpărat sistemul de operare DOS (Disk Operating System) și l-a adaptat cerințelor IBM creând MS-DOS. Sistemul a câștigat o poziție dominantă pe piață datorită deciziei lui Gates de a-l direcționa către companiile producătoare de hardware, și nu către utilizatorii finali.

În 1981 au apărut primele calculatoare personale IBM, costând 2880 USD, incluzând 64 KB de memorie și un floppy disk. Piața crește rapid, apărând și alte modele care foloseau aceeași arhitectură, dat fiind că IBM a decis să o facă publică. Vasta majoritate a acestor calculatoare rulau sub MS-DOS.

Doug Engelbart a realizat interfața grafică pentru utilizatori - GUI, care a fost apoi încoporată în echipamentele construite de Xerox PARC. Steve Jobs, co-inventatorul calculatorului Apple, a transformat GUI într-un succes comercial remarcabil prin lansarea Apple Macintosh în 1984 (dotat cu mouse) și direcționarea acestuia către utilizatorii cu cunoștințe sau aptitudini tehnice minimale.

Succesul lui Macintosh a inspirat Microsoft în elaborarea Windows, conceput inițial ca o interfață grafică peste MS-DOS. În 1995 a apărut prima versiune autonomă de Windows, urmând apoi Windows 98. Windows NT, o versiune complet nouă de Windows, s-a impus treptat pe piața utilizatorilor corporativi, fiind orientat spre servere și oferind posibilități avansate de administrare. A fost urmat de Windows 2000 și Windows 2003 Server. Windows 98 a fost urmat de Windows Me, care a adus îmbunătățiri majore în modul de pornire a sistemului de operare (prin eliminarea completă a modului real de adresare al MS-DOS-ului, folosit până atunci pentru lansarea Windows-urilor). În 2001 arhitectura NT a migrat către utilizatorii individuali prin lansarea lui Windows XP (eXPerience). Prin Windows Vista, în prezent, se dorește păstrarea acestor orientări simultane atât spre partea de server cât și spre partea de utilizator.În această perioadă sistemele UNIX și Linux încep să câștige teren. Este dezvoltată o primă interfață grafică pentru ele – X Windows, urmată de versiuni mai complexe.

Datorită expansiunii rețelelor de calculatoare personale apar sisteme de operare specializate (sisteme de operare distribuite și sisteme de operare de rețea).

1.3 Sisteme Windows versus sisteme Linux?

În momentul de față, pe majoritatea arhitecturilor PC, de o popularitate sporită se bucură sistemele de operare Microsoft Windows și distribuțiile Linux. Sistemele Apple Macintosh au migrat de la o arhitectura PowerPC spre arhitectura Intel de relativ puțin timp, intrând astfel în competiție directă cu Windows și Linux. Întrucât primele două sisteme sunt concurente directe pe piață, apar întrebări firești legate de asemănările și deosebirile dintre ele, care sunt punctele tari și punctele slabe ale fiecăruia și, în ultimă instanță, care dintre ele este cel mai bun. În realitate însă, a compara aceste două clase de sisteme este un lucru foarte dificil, date fiind diferențele fundamentale de concept și realizare, iar mai mult, a trage concluzia că unul dintre sisteme este implicit mai bun sau mai slab decat celălalt este imposibil.

Ambele sistemele oferă performanțe avansate, un mediu stabil de operare și au un domeniu foarte bogat de aplicații disponibile și concepute special pentru acestea. Dar pentru că filozofiile din spatele manierei în care sistemele au fost proiectate și au evoluat sunt diferite, concluzia finală depinde în mare măsură de scopul în care sistemul va fi folosit, gradul de pregătire al utilizatorilor, configurația hardware și mulți alți factori subiectivi.

Diferența dintre Windows și Linux este și una de ordin semantic. Microsoft Windows desemnează un produs bine delimitat, având asociate niște specificații, pe baza cărora există un preț pe piață. Linux, în schimb, nu reprezintă un produs bine definit, termenul având mai multe înțelesuri. Tehnic vorbind, Linux este de fapt numai numele kernel-ului (nucleului) sistemului de operare. Ca fapt divers, Linux este o marca inregistrata a lui Linus Torvalds, inițiatorului proiectului kernel-ului de Linux. În prezent, este și coordonatorul proiectului. Kernel-ul nu depinde de restul sistemului, așa cum nici restul pachetelor care compun sistemul nu depind în mod necesar de kernel, cu câteva excepții notabile. Cu toate acestea, majoritatea sistemelor au ajuns să fie construite pe serie comună de pachete, bazată pe proiectul GNU (http://en.wikipedia.org/wiki/GNU), întreg sistemul fiind pus laolaltă în ceea ce se cheamă o distribuție: o colecție de pachete, împreună cu kernel-ul Linux, de întreținerea și actualizarea careia se ocupă o anumită organizație sau comunitate. Astfel, printr-un abuz de limbaj unanim acceptat, aceste clase de sisteme de operare sunt denumite distribuții Linux (sau GNU-Linux), dată fiind legătura puternică cu kernel-ul și pachetele de bază construite peste el.

Microsoft a lansat la început Windows ca pe o colecție de aplicații MS-DOS, rulate într-un mediu grafic interactiv, care să ușureze munca utilizatorului de a lucra cu sistemul. Interfața intuitivă și prietenoasă, suita bogată de aplicații preinstalate, suportul tehnic dedicat, toate acestea au promovat Windows-ul ca fiind alegerea ideală pentru sistemul desktop de uz personal sau pentru birou. Acest atu a fost pastrat si la edițiile ulterioare de Windows, unde s-au facut progrese semnificative în direcția performanțelor: s-a desprins de MS-DOS, capătând un kernel modern, cu suport de multitasking și mai mulți utilizatori, interfața grafică a devenit mai bogată și mai atragatoare și nu în ultimul rând necesitatea de cunoștinte tehnice din partea utilizatorilor s-a diminuat de la ediție la ediție. Politicile de marketing ale Microsoft sunt de a face acest sistem accesibil unui număr cât mai vast de clienți, cu un efort minim de configurări ulterioare și o experiență de utilizare cât mai plăcută, îndelungată și, eventual, suficientă pentru nevoile oricărui utilizator.

Primele sisteme având la bază Linux, pe de altă parte, au fost realizate în mediul academic. Dorind o alternativă gratuită și liberă pentru sistemele UNIX comerciale de la acea vreme, o comunitate de entuziaști s-a adunat în jurul lui Linus Torvalds pentru a ajuta la îmbunătățirea și extinderea kernel-ului creat de acesta inițial doar ca pe un proiect la facultate. Mai departe, o multitudine de proiecte software având la bază acest kernel au fost înființate, cu numărul și importanța acestora crescând de la an la an. Spre deosebire de cazul Windows-ului, software-ul pe Linux nu a fost scris neapărat având în minte ușurința de utilizare și accesibilitatea, deși aceasta este tendința din prezent. Fiind vorba îndeosebi de mediul academic, majoritatea utilizatorilor aveau cunoștințe avansate de IT și apreciau la un sistem stabilitatea, flexibilitatea și performanța, mai degrabă decât o interfață simplă și intuitivă. Datorită acestor puternice avantaje, Linux (ca majoritatea sistemelor UNIX, dealtfel) a avut un succes exploziv pe servere, și s-a impus într-o măsură mult mai mică pe desktop, unde supremația este încă deținută de către sistemele de operare de la Microsoft. În momentul de față, o bună parte din infrastructura Internet-ului este ținută pe Linux.

Principala diferență dintre Linux și multe alte sisteme de operare actuale, mai mult sau mai puțin răspândite, este faptul că atât componentele (aplicațiile) cât și kernel-ul său sunt gratuite și, mai mult decât atât, open-source, ceea ce înseamna că o comunitate vastă de utilizatori și ingineri software au acces la ele, pentru a le utiliza sau a le aduce noi îmbunătățiri. Acestă practică nu este nouă, ci datează de la începuturile Linux-ului, chiar de la varianta incipientă a acestuia, Minix, un sistem de operare ce a fost conceput de la bun început pentru a fi gratuit și accesibil tuturor. Conceptul de licență sub care funcționează Linux, alături de aplicațiile sale, a primit denumrea de "*copyleft*" care, contrar copyright-ului, oferă oricui dreptul de acces, distribuție și modificare a unei creații, atâta timp cât păstrează aceleași caracteristici ale "*copyleft*"-ului. La polul opus, majoritatea sistemelor de operare ce nu aparțin familiei Linux și,

cu preponderență cele comerciale, se supun în continuare legii copyright-ului şi împiedică intervenția utilizatorilor asupra codului sursă (*closed source*).

Comunitatea Open Source este una foarte dinamică, iar în acest moment distribuțiile de Linux beneficiază din plin de acest lucru. Până acum câțiva ani, sistemele Linux necesitau cunoştințe avansate de operare, din motive legate de instrumente grafice de administrare limitate sau lipsa driverelor din partea producătorilor de hardware. Mai mult, accesul limitat la Internet priva utilizatorul de rând de accesul la documentație (deşi vastă, în mare parte online), esențială pentru încurajarea utilizatorilor de a experimenta cu un nou sistem de operare, în contextul în care nu existau campanii de marketing ca în cazul produselor comerciale.

Lucrurile au început totuşi să se schimbe într-un mod radical în momentul în care Internet-ul a devenit o resursă din ce în ce mai accesibilă şi o serie de companii au remarcat potențialul pe care Linux îl avea şi şi-au orientat domeniul de activitate către acesta. Interesul crescut a încurajat comunitățile să depună un efort mai mare pentru a face aplicațiile mai accesibile. Robustețea şi performanțele superioare au fost îmbinate cu o interfață bogată şi o experiență vizuală comparabilă (dacă nu superioară - vezi proiectele Compiz şi Beryl) cu cea de pe Windows. Suportul din partea producătorilor de hardware a crescut la rândul său, cu implicări foarte puternice din partea multor ca Novell, Canonical, sau Dell. Astfel, desktop-ul Linux devine o opțiune atrăgătoare, anii următori anunțându-se extrem de promițători pentru ascensiunea mişcării Open Source.

Există reticențe cu privire la încurajarea răspândirii programelor open-source din moment ce o astfel de libertate a tuturor utilizatorilor este dificil de controlat şi poate crea probleme de interoperabilitate. Într-adevăr, unificarea pe care o oferă sistemele de operare comerciale, concepute global şi, totodată, compact, impune şi uniformizarea aplicațiilor scrise pentru a rula pe ele, în timp ce varietatea deja mare şi în continuă creştere a sistemelor open-source conduce la stabilirea unei multitudini de direcții de dezvoltare între care se împarte software-ul conceput pentru toate acestea. Pe de altă parte, comunitatea de utilizatori ai programelor open-source este în continuă creştere, fenomen ce poate reprezenta o amenințare către monopolul indus de către Microsoft, spre exemplu. În ultimă instanță, diversitatea poate fi greu de controlat dar totodată, un foarte mare segment al programatorilor din lume lucrează la întreținerea şi aducerea de inovații / îmbunătățirea acestor proiecte.

Cea mai mare comunitate de programatori ce dezvoltă aplicații open-source pentru absolut toate platformele şi sistemele de operare, se găseşte la "http://sourceforge.net". Puțin cunoscut este faptul că şi Microsoft a încurajat dezvoltarea unei comunități, numită "Codeplex", axată pe dezvoltarea aplicațiilor open-source folosind, bineînțeles, tehnologiile Microsoft, precum .NET Framework, Windows Presentation Foundation, XNA, etc, aplicațiile fiind concepute pentru sistemele de operare de la Microsoft, cu preponderență Window XP şi Windows Vista. Siteul acestei comunități poate fi accesat la adresa http://www.codeplex.com.

1.4 Studii de caz

1.4.1 MAC OS

MAC OS reprezintă o familie de sisteme de operare concepute special pentru calculatoarele Macintosh („Mac"-uri) produse de firma Apple. Introduse inițial în 1984, sistemele de operare pentru Macintosh au cunoscut o evoluție spectaculoasă până în prezent, menținându-se drept concurent pentru Microsoft Windows atât prin performanțele sistemului propriu-zis cât şi prin versatilitatea şi utilitatea aplicațiilor pe care le dețin.

Calculatoarele Macintosh au implementat, de la lansare, o arhitectură hardware de tip Power PC, migrând mai apoi către o arhitectură Intel, moment în care sistemele de operare pentru Mac-uri au început să reprezinte o concurență pentru Linux şi Windows.

Sistemul de operare a fost radical schimbat la lansarea versiunii 10.0, prin rescrierea nucleului. Această schimbare este reflectată şi în modificarea denumirii sistemului de operare din MAC OS în MAC OS X. Adăugarea literei X subliniază trecerea de la tehnologia MAC OS nanokernel, la tehnologie derivată din UNIX.

În prezent, varianta curentă de MAC OS X este 10.4, urmată de varianta 10.5 („Leopard") în octombrie 2007. Prin sistemul lor de operare şi prin aplicaţiile pe care acesta le pune la dispoziţia utilizatorilor, Macintosh-urile excelează în domeniul prelucrării grafice dar totodată continuă să aducă inovaţii orientate spre nevoile utilizatorului individual.

Figura 1-2 Imagine din MAC OS X 10.4

1.4.2 Virtualizare. Rularea unui SO dintr-o maşină virtuală

O maşină virtuală este, în esenţă, o aplicaţie ce creează un mediu de execuţie pentru rularea unui nou sistem de operare, folosind resursele gestionate de sistemul de operare gazdă, prin intermediul mecanismelor oferite de maşina virtuală. O caracteristică particulară a maşinii virtuale este capacitatea acesteia de a izola procesele ce rulează sistemul de operare din interiorul său de cele ale sistemului de operare gazdă.

Resursele alocate de către maşina virtuală sunt exprimate prin intermediul mai multor aspecte: un procentaj dedicat din memoria RAM, un spaţiu fizic de stocare de pe hard-disk-ul calculatorului gazdă (alocat în prealabil sau în timp real) pe care maşina virtuală îl va folosi ca pe un hard-disk propriu şi pe care va putea instala chiar un sistem de fişiere complet diferit de cel pe care a fost alocat spaţiul, accesul la diferite dispozitive periferice (maşina virtuală se ocupă de medierea semnalelor dispozitivelor hardware spre şi dinspre sistemul de operare gazdă şi cel „găzduit"). Accesul celui de-al doilea sistem de operare la procesor se face prin intermediul proceselor maşinii virtuale, care vor rula alături de celelalte aplicaţii şi vor împărţi aceleaşi resurse.

Rularea sistemelor de operare într-o maşină virtuală oferă şi avantajul unei porniri a sistemului mai rapide în lipsa necesităţii de a realiza teste şi iniţializări hardware.

Cuvinte cheie

- API
- biblioteci
- GUI
- Linux
- MAC OS
- nucleu (kernel)
- open-source
- portabilitate
- programe de bază
- sistem de operare
- virtualizare
- Windows

Întrebări

Care dintre următoarele aplicaţii poate fi privită ca o componentă a sistemului de operare?

- ❑ procesor de text
- ❑ player pentru fişiere video
- ❑ interpretor de comenzi
- ❑ client de email

Care dintre următoarele nu reprezintă o resursă partajată în cazul virtualizării?

- ❑ RAM
- ❑ hard-disk
- ❑ procesor
- ❑ BIOS

Care dintre următorii este autorul unui limbaj de programare?

- ❑ Larry Wall
- ❑ Linus Torvalds
- ❑ William Henry Gates III
- ❑ Andrew S. Tanenbaum

Care dintre următoarele nu este o funcţie a sistemului de operare?

- ❑ controlul accesului la memorie
- ❑ căutarea si eliminarea programelor virus
- ❑ asigurarea comunicării între procese
- ❑ organizarea fişierelor în directoare

2 Instalarea Linux. Configurări de bază

UNIX is user-friendly. It's just very selective about who its friends are.

Ce se învaţă din acest capitol?

- Ce este Linux?
- Distribuţii Linux
- Partiţionarea discului; pregătirea pentru instalare
- Instalarea Kubuntu
- Interfaţa cu utilizatorul; interfaţa în linia de comandă
- Autentificarea în sistem
- Gestiunea utilizatorilor

2.1 Linux. Distribuţii de Linux

Linux reprezintă o întreagă clasă de sisteme de operare şi nu un sistem de operare concret. Conceptul de Linux ca şi colecţie de programe aflate sub o licenţă GPL poate fi destul de uşor asociat cu Linux Kernel datorită modului în care acesta este organizat şi dezvoltat.

Proiectele *open source*, deşi sunt dezvoltate într-o manieră colaborativă, sunt produse independent unele de altele. Datorită licenţelor sub care se afla aceste proiecte, licenţe ce permit redistribuirea codului sursa, este posibilă apariţia unor proiecte mai mari, care să adune la un loc programe dezvoltate independent şi să le facă disponibile la un loc sub forma unei distribuţii.

Prin urmare, o distribuţie de Linux reprezintă o colecţie particulară de programe, unele din ele modificate special pentru acea distribuţie, care sunt combinate cu o versiune de Kernel, de multe ori şi aceasta modificată pentru acea distribuţie, pentru a crea un sistem de operare.

Distribuţiile de Linux au căpătat de-a lungul timpului cele mai variate forme şi destinaţii. Unele sunt destinate funcţionării pe servere, altele funcţionării pe sisteme desktop, iar altele sunt optimizate pentru a ocupa cât mai puţine resurse putând astfel fi folosite în sistemele *embedded*.

2.1.1 Principalele distribuţii de Linux

În momentul de faţă există peste 300 de distribuţii active (care sunt dezvoltate în continuare). Dintre ele doar puţin peste 10 sunt populare fiind utilizate pentru scopuri generale.

De cele mai multe ori o distribuţie nu este dezvoltată de la zero, punctul de plecare constituindu-l un anumit stadiu al unei alte distribuţii. Noua distribuţie poate moşteni de la cea din care este dezvoltată diverse elemente, cum ar fi modul în care se instalează aplicaţii noi, locul în care se găsesc diverse fişiere de configurare, felul în care arată aceste fişiere, etc.

Principalele distribuții sunt:

Nume	Anul apariției	Derivată din
Slackware	1993	SLS
Debian	1993	-
Ubuntu	2004	Debian
Kubuntu	2005	Ubuntu
Red Hat	1994	-
Fedora	2003	Red Hat
Mandriva	1998	Red Hat
SUSE	1994	Slackware
OpenSUSE	2006	SUSE
Gentoo	1999	-
Knoppix	2003	Debian
Linspire	2001	Debian

Figura 2-1 Principalele distribuții de Linux

Popularitatea unei distribuții poate fi urmărită pe pagina web `distrowatch.com`. Aici este realizat un clasament al distribuțiilor pe baza numărului de accesări ale informațiilor despre distribuție pe pagina amintită. Chiar dacă acest clasament nu ia în considerare efortul depus pentru menținerea distribuției, el arată în mare măsură interesul manifestat de utilizatori.

Cele mai accesate 16 distribuții în ultimele șase luni, la data de 14.09.2007, sunt:

Nr. Crt	Distribuție	Număr accesări
1	Ubuntu	2530<
2	PCLinuxOS	2518>
3	openSUSE	1422<
4	Fedora	1284=
5	Sabayon	1198>
6	Mint	1012<
7	Debian	1003<
8	MEPIS	868=
9	Mandriva	748<
10	Damn Small	666<
11	Slackware	585=
12	CentOS	543<
13	Gentoo	508<
14	KNOPPIX	469<
15	Zenwalk	462<
16	Kubuntu	450<

Figura 2-2 Cele mai accesate distribuții pe distrowatch.com

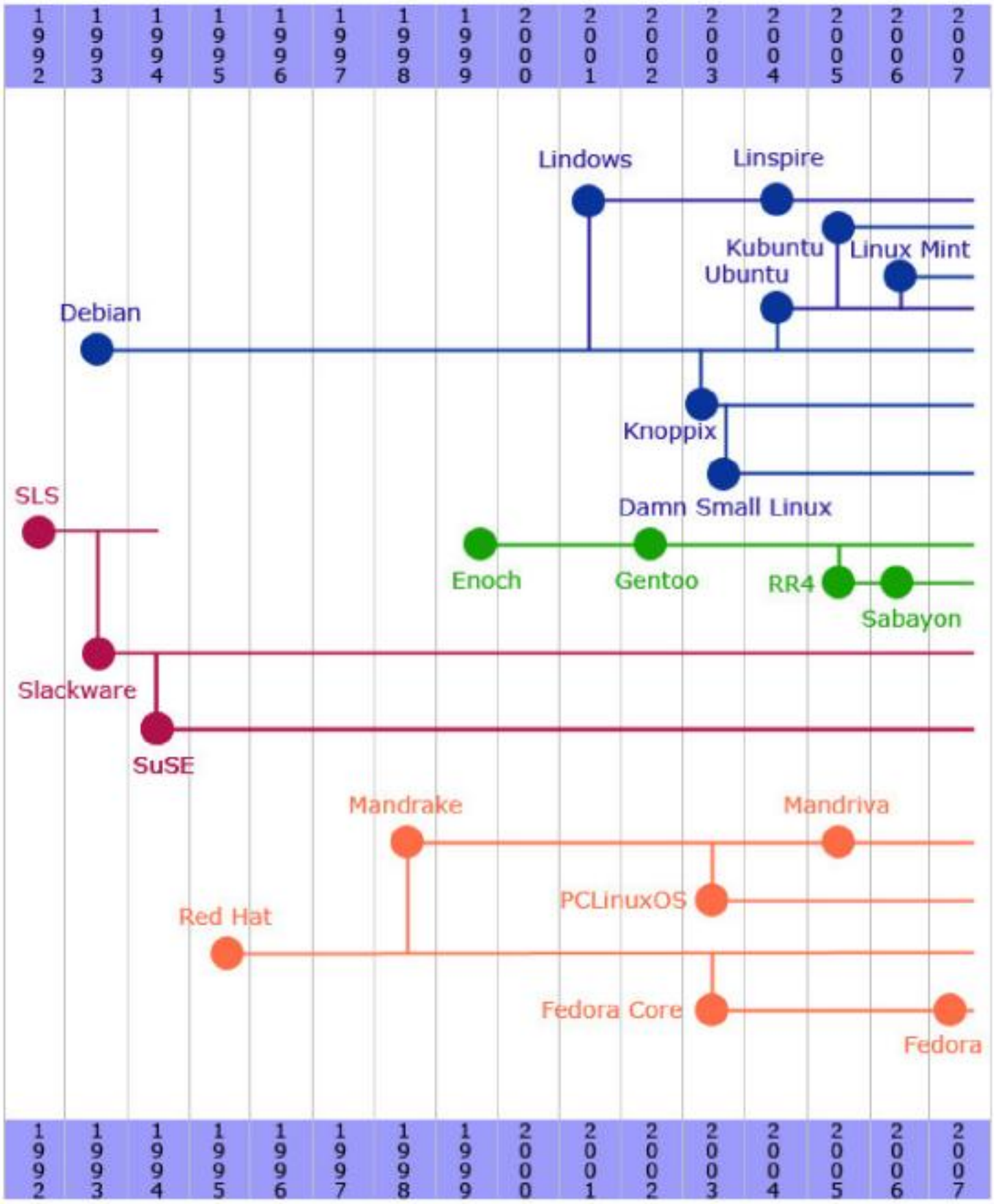


Figura 2-3 Evoluția principalelor distribuții de Linux

Pe lângă distribuțiile de uz general (pentru servere sau pentru calculatoare personale) există şi distribuții dedicate unui scop precis (*one purpuse*). De exemplu **Musix**, distribuție dedicată producerii de material multimedia, **SystemRescueCD**, distribuție LiveCD [vezi 2.3.2] specializată în repararea unui calculator pe care sistemul de operare nu se mai inițializează din motive ce țin de software, sau **BackTrack**, distribuție Live CD care oferă uneltele necesare atacării unei rețele de calculatoare.

O clasă aparte de distribuții o reprezintă cele care au ca țintă ocuparea a cât mai puțin spațiu. Un exemplu în acest sens este **DamnSmallLinux**, care reuşeşte să includă în doar 50MB o distribuție completă cu interfață grafică şi cu toate aplicațiile uzuale.

Există multe diagrame disponibile pe Internet ce arată evoluția distribuțiilor şi felul în care au derivat una din alta. O simplă căutare a şirului de caractere *„linux distro timeline"* pe `images.google.com` va oferi câteva zeci de variante ale acestei diagrame. O versiune care conține doar principalele distribuții este prezentată în figura 2.3.

2.1.2 Debian. Ubuntu

Una dintre primele distribuții apărute este **Debian**. Din 1993 şi până în prezent Această distribuție a reprezentat punctul de plecare a peste 40 de distribuții.

Printre avantajele pe care le oferă Debian se numără: suportul bun disponibil pe Internet atât pe pagina web oficială (`http://www.debian.org/support`) cât şi alte forumuri. Distribuția oferă posibilitatea de a configura sistemul de operare pe baza pachetelor, având disponibile peste 17000 de pachete. Debian oferă suportul pentru a actualiza într-un mod facil şi organizat sistemul de operare. De asemenea sunt disponibile trei variante ale distribuției, *stable*, *testing* şi *unstable*, care permit alegerea gradului de stabilitate şi noutate dorit.

Cea mai cunoscută distribuție derivată din Debian este **Ubuntu**. În momentul de față Ubuntu este şi distribuția cu cea mai mare popularitate dintre toate distribuțiile existente, fiind pe primul loc în clasamentul `distrowatch.com` de aproape doi ani. Atuul principal al acestei distribuții este faptul că este uşor de folosit pentru utilizator. Ea conține toate aplicațiile de care un utilizator are nevoie şi realizează multe dintre configurări automat. Un avantaj mare pe care îl oferă este suportul disponibil pe forumul oficial.

O schimbare importantă pe piața de calculatoare s-a produs în mai 2007 când Dell, unul dintre cel mai mari producători de echipamente IT, a anunțat că va scoate pe piață calculatoare desktop şi laptop-uri care au preinstalat Ubuntu. Este prima dată când în locul Windows-ului a fost preferată o soluție Linux.

2.1.3 Stable vs. bleeding edge

În IT există două concepte importante legate de stabilitatea, siguranța şi gradul de noutate al aplicațiilor (şi nu numai): versiunile stabile (*stable*) şi cele cu un grad de noutate ridicat (*bleding edge*). Aceste două concepte reprezintă două extreme ale modului în care este dezvoltată o aplicație.

Versiunile **stabile** (*stable*) reprezintă versiuni ale programelor care au fost testate un timp îndelungat, ale căror bug-uri au fost rezolvate, care nu prezintă probleme de incompatibilitate şi care au un grad de siguranță ridicat. Într-o versiune stabilă este destul de improbabil să apară breşe de securitate. Aceste versiuni apar rar deoarece este nevoie de timp pentru a crea stabilitatea necesară.

Dezavantajul versiunilor stabile este acela că aplicațiile nu prezintă nici un grad de noutate. De multe ori pot să dureze ani până când o nouă versiune a unei aplicații să fie introdusă într-un Linux stabil.

Versiunile stabile sunt folosite în special pentru severe datorită gradului de stabilitate şi se siguranță oferit.

Versiunile cu un **grad de noutate ridicat** (*bleeding edge*) reprezintă versiuni ale programelor în care sunt implementate cele mai noi îmbunătățiri şi au incluse ultimele inovații. Aceste versiuni apar destul de des, frecvența lor de apariție depinzând în mare măsură de mărimea comunității ce dezvoltă aplicația şi de îmbunătățirile aduse.

Dezavantajul versiunilor cu grad de noutate ridicat este acela că prezintă un grad de stabilitate şi de siguranță scăzut. Datorită lipsei de timp alocat testării extensive a aplicațiilor, acestea prezintă deseori găuri de securitate sau probleme de stabilitate.

Versiunile cu grad de noutate ridicat sunt recomandate celor care doresc să aibă cele mai noi tehnologii implementate cu orice cost. Ele nu sunt recomandate pentru servere, ci pentru calculatoare unde stabilitatea şi siguranţa nu sunt elemente cheie.

Între cele două tipuri de aplicaţi prezentate există destule niveluri de compromis. Alegerea unui astfel de nivel de compromis în ceea ce priveşte un sistem de operare (mai aproape de stabilitate sau mai aproape de cele mai noi tehnologii) trebuie făcută ţinând cont de destinaţia calculatorului/serverului pe care va fi instalat şi de gradul de securitate şi de stabilitate pe care trebuie să îl prezinte.

2.2 Partiţii

2.2.1 Terminologie

O partiţie reprezintă o diviziune logică a unui hard-disk. Prin partiţionare un hard-disk este împărţit în două sau mai multe partiţii.

Atunci când pe un hard-disk se găseşte o singură partiţie care ocupă întreg spaţiul disponibil, hard-disk-ul se numeşte nepartiţionat.

Partiţionarea unui hard-disk îl face pe acesta să aibă aceeaşi comportare ca mai multe hard-disk-uri independente nepartiţionate. Diferenţa între partiţii şi hard-disk-uri reale constă în faptul că partiţiile de pe un hard-disk folosesc aceleaşi resurse hardware (şi astfel, dacă discul se strică, toate partiţiile create pe el devin inaccesibile). În plus partiţiile pot fi redimensionate după ce au fost create, în schimb disk-urile au o dimensiune fixă.

Într-o partiţie se poate crea un sistem de fişiere pentru stocarea fişierelor sau partiţia poare fi utilizată în scopuri speciale, cum ar fi de exemplu partiţiile de swap.

Sistemul de operare foloseşte pentru stocarea datelor temporare memoria RAM. Aceasta are avantajul unui timp de răspuns şi al unei viteze de citire foarte mari, însă este limitata din punctul de vedere al capacităţii (sistemele actuale nu au, în mod normal, mai mult de 2GB de memorie RAM).

Atunci când întregul conţinut al memoriei RAM este umplut cu date, există două alternative. Prima din ele constă în nealocarea de memorie RAM pentru programele care cer acest lucru. Ca urmare programul respectiv nu va putea funcţiona şi va genera erori. Atunci când programul în cauză este unul vital pentru funcţionarea sistemului de operare, funcţionarea întregului sistem de operare poate deveni instabilă.

A doua variantă este aceea în care datele care ar trebui stocate în memoria RAM sunt stocate într-o alta memorie (pe hard-disk sau pe o memorie pe USB). Deoarece memoria RAM este cea mai rapidă memorie din sistem, utilizarea unui alt tip de memorie va atrage după sine penalizări de performanţă. Această soluţie se numeşte **swapping** datorită modului în care funcţionează: dacă memoria RAM este plină şi apar cereri pentru alocarea de noi zone de memorie, o parte din datele aflate deja în memoria RAM sunt copiate pe un alt tip de memorie, în acest fel eliberându-se memorie RAM. În momentul în care datele copiate anterior pe un alt tip de memorie sunt cerute de programe, ele sunt copiate înapoi în RAM (daca nu este suficient loc în acel moment se va elibera o parte din RAM copiind alte date din RAM pe suportul de memorie).

Cea mai uzuală soluţie de memorie folosită pentru swap este hard-disk-ul. Sistemele de operare de tip Linux folosesc o partiţie creată special în acest sens, numită **partiţie de swap**. Această partiţie are în general dimensiuni apropiate de cele ale memoriei RAM (de ordinul 1-2GB). Windows-ul foloseşte pentru swap câteva fişiere a căror dimensiune se poate extinde sau micşora funcţie de necesităţi.

Partiţiile de swap nu folosesc un sistem de fişiere ca NTFS, FAT32 sau ext3. Ele au însă metode de a organiza datele şi de a le face uşor de accesat.

Avantajele utilizării de partiții sunt următoarele:

- Posibilitatea instalării mai multor sisteme de operare pe același hard-disk putând alege, la pornirea sistemului de calcul, care din ele va fi folosit
- Posibilitatea de a separa datele personale de datele folosite de sistemul de operare și de programe. În cazul în care sistemul de operare nu mai poate fi pornit și este necesară reinstalarea lui, dacă datele personale ale utilizatorilor se găsesc pe o altă partiție, ele nu vor fi afectate în nici un fel
- Este mai eficient să existe mai multe partiții mici decât o singură partiție mare, deoarece timpul de căutare al datelor este mai bun în cazul unei partiții mici decât în cazul unei partiții mari
- Utilizarea aceluiași spațiu de swap între mai multe sisteme de operare de tip Linux instalate pe același sistem de calcul (folosind aceeași partiție de swap pentru toate)

2.2.2 Tipuri de partiții

Master boot record (MBR) [vezi 5.2.1.1] este denumirea dată primului sector fizic de pe un hard-disk. Acest sector are 512 octeți. MBR conține informații importante despre modul în care este partiționat hard-disk-ul. Dacă acest sector este șters sau suprascris, atunci toate informațiile de pe hard-disk se pierd.

În cadrul MBR se găsesc informații despre fiecare partiție de pe hard-disk. Aceste informații sunt stocate într-o tabelă de partiții primare care are doar patru intrări. Prin urmare MBR-ul poate reține informații despre doar patru partiții primare. Una dintre aceste patru partiții va fi marcată ca activă și de pe ea se va încerca pornirea sistemului de operare.

O **partiție primară** este o partiție care conține un singur sistem de fișiere. Pentru a elimina limitarea pe care o prezintă MBR-ul (de a putea reține doar patru partiții primare) a fost creat un al doilea tip de partiții, numit **partiție extinsă**. Acest tip de partiție ocupa tot o intrare din tabela de partiții din MBR însă poate conține la rândul său mai multe **partiții logice**, având fiecare asociat un sistem de fișiere. Pe un hard-disk se poate găsi o singură partiție extinsă. Numărul de partiții logice incluse într-o partiție extinsă nu este limitat decât de dimensiunile partiției extinse.

Unele sisteme de operare trebuie instalate pe o partiție primară pentru a putea porni. În această categorie intră MS-DOS și toate versiunile de Windows care depind de el (Windows 3.X, Windows 95, Windows 98, Windows Millenium), precum și BSD.

2.3 Instalare Kubuntu

2.3.1 Alegerea tipului de distribuție: grafic, alternate, server

Pentru fiecare versiune de Ubuntu/Kubuntu există trei variante de instalare. Instalarea **server** este mai apropiata de Debian și de stabilitatea acestuia. De aceea este recomandată pentru calculatoare ce trebuie să ofere stabilitate și siguranță.

Instalarea **grafică** (cea mai uzuală) permite utilizatorului să instaleze varianta desktop a sistemului de operare folosind un asistent în mod grafic. Aceasta are nevoie de minim 256MB de RAM pentru a putea funcționa.

Instalarea **alternate** va permite utilizatorului să instaleze tot varianta desktop, însă asistentul de instalare nu va mai fi unul în mod grafic ci unul în mod text, bazat pe asistentul de instalare al distribuției Debian. Din acest motiv varianta alternate poate fi instalată și pe sisteme cu mai puțin de 256MB de RAM. Un alt avantaj al variantei alternate este acela că permitea realizarea unei partiționări a hard-disk-ului mai avansată, oferind suport pentru utilizarea RAID

sau LVM (ambele tehnologii permit realizarea de partiții care includ spațiul de pe mai multe hard-disk-uri).

2.3.2 LiveCD

Pentru Ubuntu/Kubuntu 7.04 varianta grafică de instalare este disponibilă sub forma unui LiveCD

LiveCD este numele generic dat unei clase de distribuții Linux care pot porni și rula fără să aibă nevoie de prezența unui hard-disk.

De obicei un LiveCD este stocat pe un mediu boot-abil (cum ar fi un CD, de la care îi vine și numele, un DVD, o memorie USB, un floppy-disk, etc). Termenul de *Live* vine de la faptul că pe un singur disc se găsesc toate fișierele necesare funcționării unui sistem de operare, fără a fi necesară copierea unui fișier pe hard-disk înaintea pornirii sistemului de operare.

Astfel un LiveCD nu modifică în nici un fel sistemul de operare instalat pe hard-disk, doar dacă utilizatorul dorește în mod expres acest lucru. Un sistem de operare prezent pe un LiveCD este totuși unul normal, astfel încât are nevoie de existența unui hard-disk pentru a putea funcționa. Pentru a rezolva această problemă un Live CD folosește pe post de hard-disk o porțiune din memoria RAM, creând un *ramdisk*. Toate datele care se scriu în *ramdisk* sunt accesibile pe toată durata în care sistemul de operare rulează. Atunci când acesta este oprit, datele din RAM se pierd și implicit tot conținutul *ramdisk*-ului este șters. De aceea, după oprirea unui LiveCD, sistemul de calcul revine la starea anterioară pornirii LiveCD-ului.

Utilizarea unei porțiuni din memoria RAM pe post de hard-disk oferă avantajul unui timp de acces și a unei rate de transfer mai bune decât în cazul unui disk. Dezavantajul este acela că aplicațiile vor avea disponibilă o cantitate mai mica de RAM. Pentru a putea rula în condiții normale un LiveCD are nevoie de 512MB de memorie RAM, minimul necesar fiind 256MB.

Așadar un LiveCD oferă posibilitatea de a putea testa și opera un sistem de operare în condiții de deplină siguranță pentru software-ul instalat pe sistemul de calcul.

2.3.3 Instalare Kubuntu

Kubuntu este o distribuție derivată din Ubuntu, care folosește *desktop environment*-ul [vezi 12.1.2] KDE în loc de GNOME. Kubuntu este parte a proiectului Ubuntu și folosește aceeași arhitectură, aceleași *repository*-uri și aceleași pachete ca și Ubuntu.

Pentru a instala Kubuntu pe hard-disk este necesară crearea a cel puțin două partiții: una pe care va fi instalat sistemul de operare și una de swap. Acest lucru poate fi realizat înaintea instalării propriu-zise (folosind programe specializate în administrarea partițiilor cum ar fi fdisk, GParted, PQMagic, etc) sau poate fi realizat în timpul instalării folosind utilitarul de partiționare pus la dispoziție în acest sens de programul asistent de instalare.

Pentru a putea fi instalat Kubuntu are nevoie de minim 256MB de memorie RAM și 4GB spațiu liber pe disc.

Instalarea Kubuntu 7.04 (varianta desktop) începe prin introducerea LiveCD-ului în unitatea optică și repornirea calculatorului. În cazul în care BIOS-ul nu este configurat să inițializeze sistemul de operare de pe unitatea optică, trebuie realizată această configurare.

După ce calculatorul a inițializat sistemul de operare de pe CD, va fi afișat un ecran ce permite alegerea modului în care va fi folosit LiveCD-ul:

Figura 2-4 Algerea modului în care va fi folosit LiveCD-ul

Prima opțiune din lista va porni Kubuntu folosind configurările implicite. A doua opțiune va porni sistemul de operare folosind o rezoluție video mai mică decât cea standard (640x480). *Memory test* va porni un test al memoriei RAM util pentru depanarea problemelor hardware, atunci când memoria RAM este suspectată că ar fi defectă

După ce este aleasă prima opțiune sistemul de operare va fi inițializat. Desktp-ul implicit din Kubuntu arată astfel:

Figura 2-5 Desktop-ul implicit din Kubuntu

În acest moment Kbuntu poate fi folosit ca LiveCD. Pentru a îl instala pe hard-disk, se va accesa pictograma *install* de pe desktop. Rezultatul va fi pornirea unui asistent de instalare:

Figura 2-6 Alegerea limbii folosită de asistentul de instalare

Prima etapă constă în alegere limbii folosită de asistentul de instalare. Etapa următoare permite alegerea locului în care se găseşte utilizatorul. Această configurare are efect asupra fusului orar ce va fi stabilit pentru calculator.

Figura 2-7 Alegerea locului în care se află utilizatorul

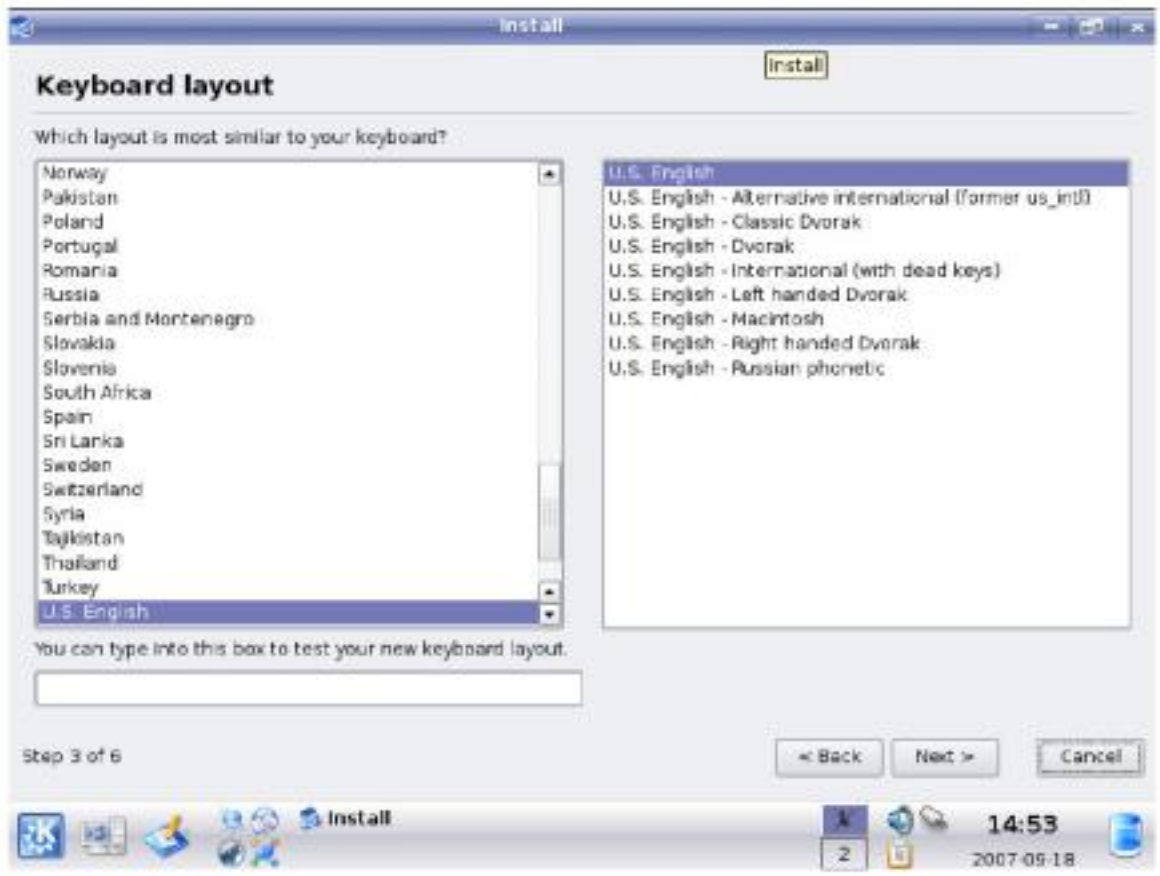


Figura 2-8 Alegerea felului în care sunt amplasate tastele pe tastatură

Pasul următor este reprezentat de alegerea felului în care sunt amplasate tastele pe tastatură (*keyboard layout*). După ce se realizează această configurare urmează pasul cel mai important din cadrul instalării: partiţionarea hard-disk-ului.

Figura 2-9 Partiţionarea manuală a hard-disk-ului

Discul poate fi şters integral, caz în care datele deja existente vor fi şterse, sau poate fi partiţionat manual, alegând dimensiunea partiţiilor şi tipul lor. În continuare va fi prezentată partiţionarea manuală.

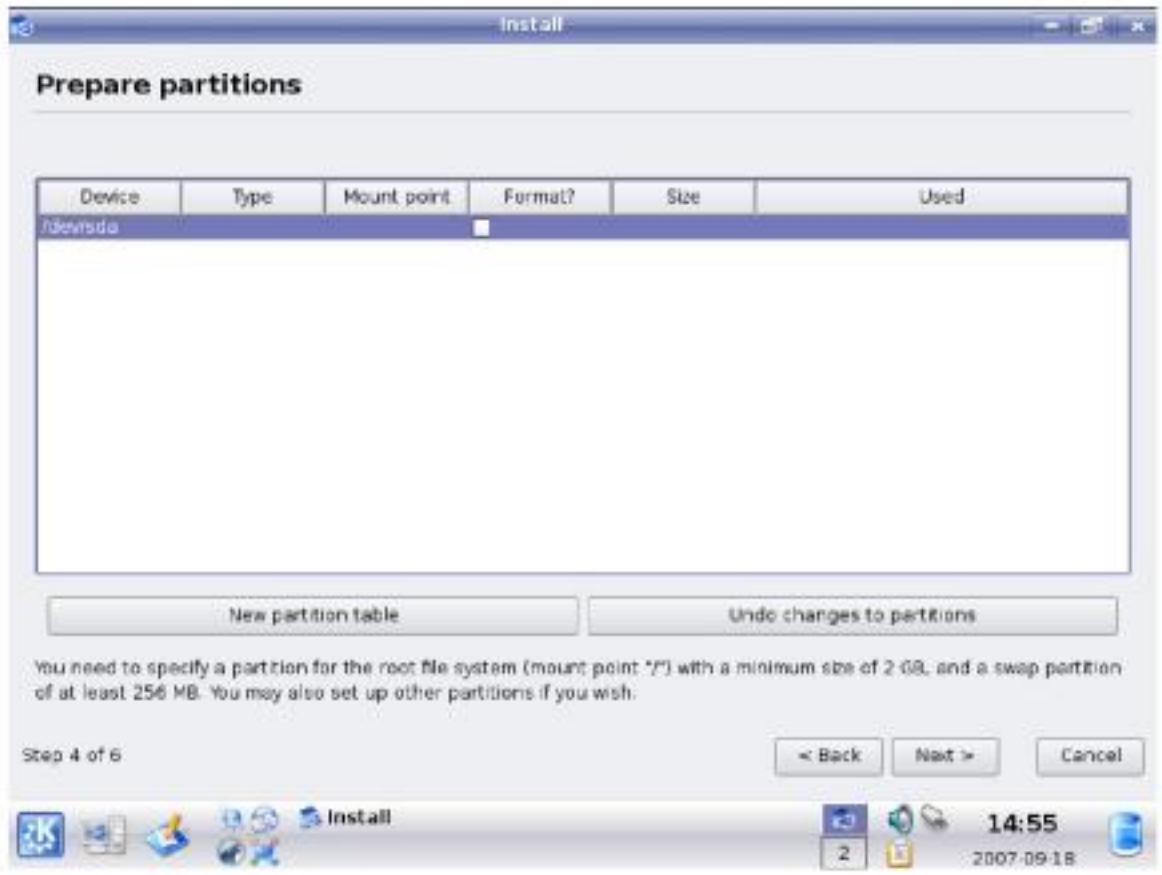


Figura 2-10 Dacă discul este gol, el trebuie partiționat

În cazul în care discul este gol (nepartiționat), se va crea o nouă tabelă de partiții apăsând butonul *New Partition table*. Rezultatul va fi afişarea tabelei de partiții:

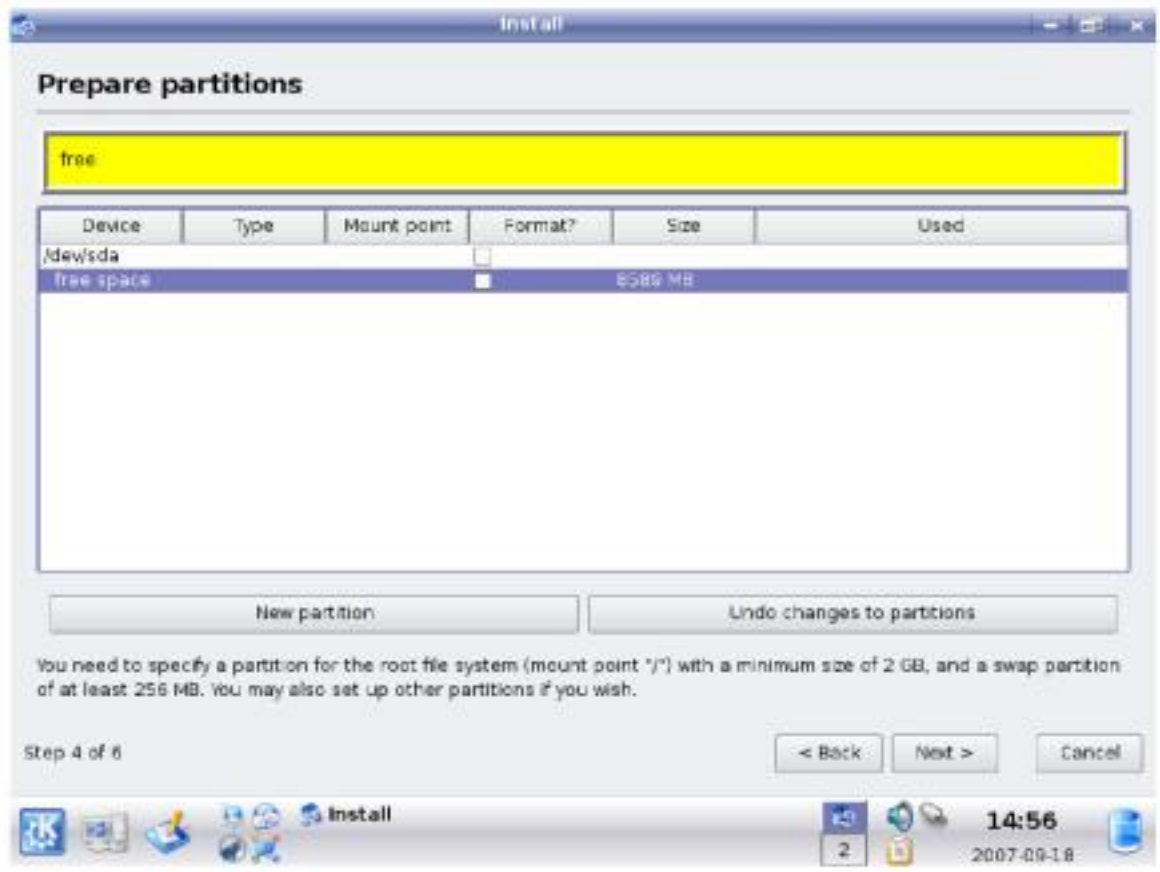


Figura 2-11 Tabela de partiții – inițial discul este gol

În cazul în care exista deja partiții pe disc, acestea vor fi afişate în tabela de partiții. În continuare se presupune ca hard-disk-ul nu este partiționat şi se vor crea două partiții: una ext3 şi una de swap. Pentru aceasta se apasă butonul *New partition* din fereastra ce arata tabela de partiții:

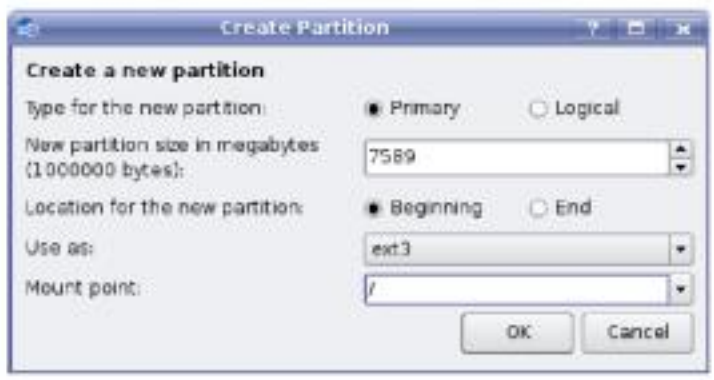


Figura 2-12 Crearea unei partiții ext3

Partiția ext3 este primară, are 7589MB, este creată la începutul spațiului liber și va fi accesată de sistemul de operare prin directorul /.

Figura 2-13 Crearea unei partiții de swap

Partiția de swap este tot primară, are 995MB și este creată la începutul spațiului liber rămas. Ea nu va putea fi accesată printr-un director, ci va fi folosită de sistemul de operare în mod transparent.

După crearea celor două partiții tabela de partiții arată în felul următor.

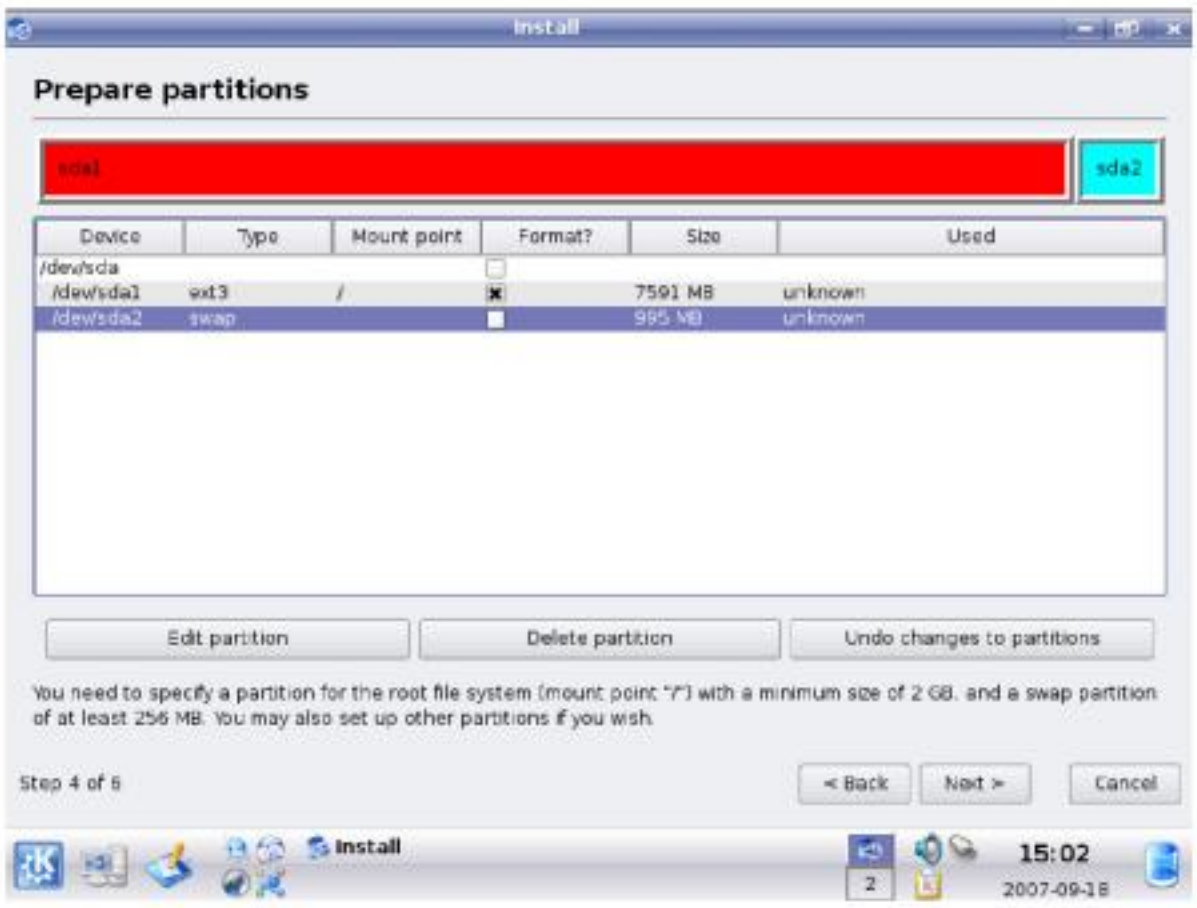


Figura 2-14 Tabela de partiții după ce au fost create cele doua partiții

După ce au fost create partițiile necesare vor fi introduse datele personale ale utilizatorului: Numele lui, numele de utilizator dorit, parola şi numele stației (*hostname*).

Figura 2-15 Configurarea datelor personale ale utilizatorului

În pasul următor (ultimul al instalării) vor putea fi revizuite configurările realizate. Apăsând pe butonul *Advanced* se va putea configura hard-disk-ul în al cărui MBR va fi instalat *bootloaderul* [vezi 5.2.3] GRUB.

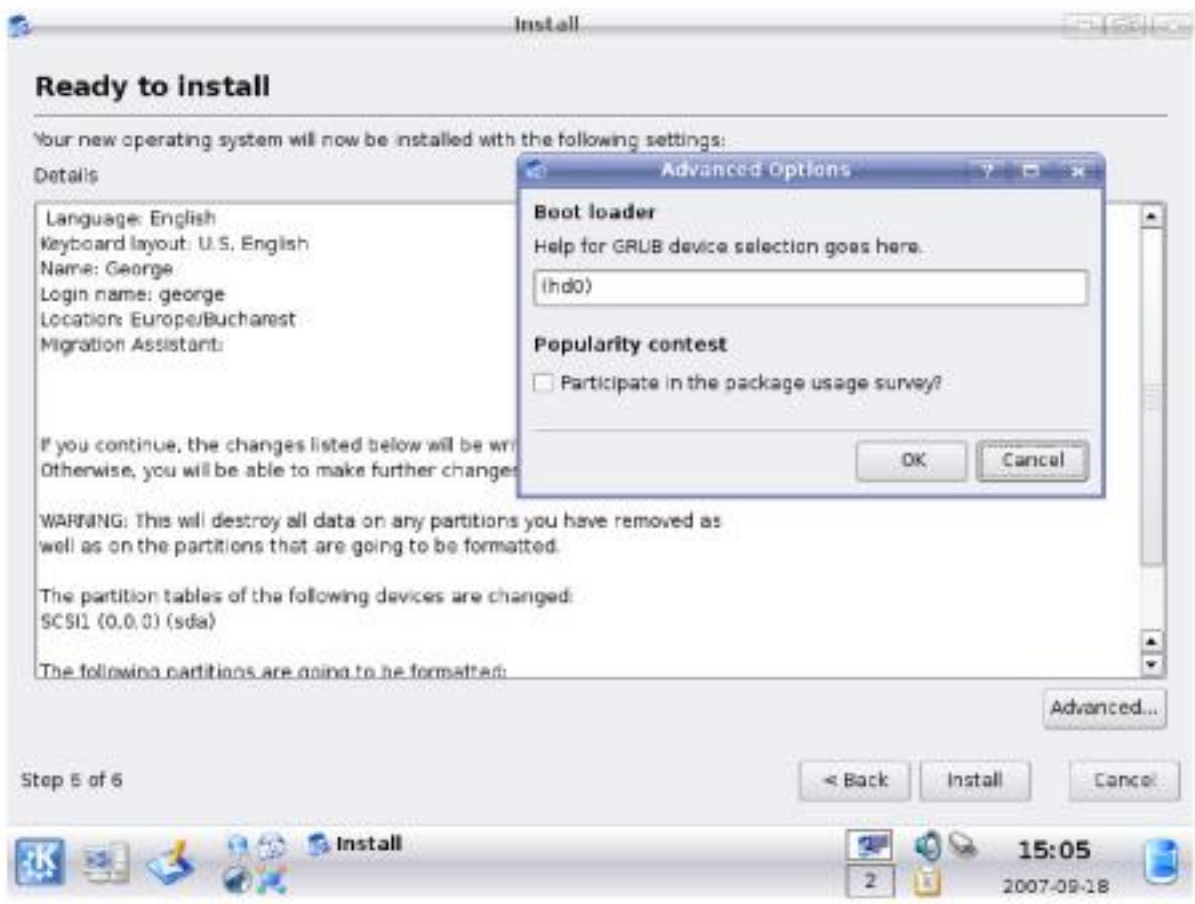


Figura 2-16 Verificarea datelor şi alegerea hard-disk-ului pe care va fi instalat Grub

Pentru a începe copierea fişierelor pe hard-disk se apară butonul *Install*. După ce instalarea a luat sfârşit va fi afişat următorul mesaj:

Figura 2-17 Mesajul care marchează insalarea cu succes a sistemului de operare

După terminarea instalării sistemul de operare va rula în continuare ca LiveCD. Pentru a porni sistemul de operare proaspăt instalat calculatorul trebuie repornit.

2.4 Interacțiunea cu sistemul de operare

2.4.1 Mediul grafic vs. CLI/TUI

Primele interfețe cu utilizatorul au fost, din motive tehnice, create pe monitoare care afişau doar caractere text. De aceea posibilitățile acestor interfețe de a crea componente grafice au fost limitate. În această categorie (interfețe care utilizează doar caractere în mod text) intră CLI (*Command Line Interface*) şi TUI (*Text User Interface*). După ce atât monitoarele cât şi plăcile video au avansat tehnologic şi au apărut afişajele grafice (cu rezoluții mai mari decât cele text), a apărut şi o nouă clasa de interfețe cu utilizatorul numită GUI (*Graphical User Interface*).

În **CLI** interacțiunea cu sistemul de operare se bazează pe comenzi scrise de la tastatură:

```
bash-2.05b$ cd /usr/portage/app-shells/bash
bash-2.05b$ ls -al
total 68
drwxr-xr-x    3 root root  4096 May 14 12:05 .
drwxr-xr-x   26 root root  4096 May 17 02:36 ..
-rw-r--r--    1 root root 13710 May  3 22:35 ChangeLog
-rw-r--r--    1 root root  2924 May 14 12:05 Manifest
-rw-r--r--    1 root root  3720 May 14 12:05 bash-2.05b-r11.ebuild
-rw-r--r--    1 root root  3516 May  2 20:05 bash-2.05b-r9.ebuild
-rw-r--r--    1 root root  5083 May  3 22:35 bash-3.0-r11.ebuild
-rw-r--r--    1 root root  4038 May 14 12:05 bash-3.0-r7.ebuild
-rw-r--r--    1 root root  3931 May 14 12:05 bash-3.0-r8.ebuild
-rw-r--r--    1 root root  4267 Mar 29 21:11 bash-3.0-r9.ebuild
drwxr-xr-x    2 root root  4096 May  3 22:35 files
-rw-r--r--    1 root root   164 Dec 29  2003 metadata.xml
bash-2.05b$ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
</pkgmetadata>
bash-2.05b$ sudo /etc/init.d/bluetooth status
Password:
 * status:  stopped
```

Figura 2-18 Command Line Interface

Avantajul acestei metode este acela că pot fi invocate comenzi complexe utilizând secvențe scurte de caractere. Pentru a realiza sarcini echivalente într-un GUI poate fi uneori nevoie şi de câteva zeci de click-uri de mouse. Un alt avantaj al CLI este rapiditatea dobândită în utilizare

după ce comenzile au fost reținute (în special dacă este folosit istoricul de comenzi: prin apăsarea tastei săgeată sus pot fi reexecutate comenzile date anterior). Luând ca exemplu configurarea unei adrese IP [vezi 7.2.1] pe o interfață de rețea, diferența de viteză între prima și a zecea oară când se realizează această configurare într-un GUI este mica, datorită numărului mare de click-uri necesar. În cazul unui CLI, folosind și istoricul de comenzi, diferența de viteză este extrem de mare.

Dezavantajul principal al CLI este curba abruptă de învățare, utilizatorul obișnuindu-se mai greu cu o comandă decât cu reținerea locului în care se află un buton.

Deși de la apariția sa a fost concurat de TUI și GUI, CLI rămâne metoda de control al sistemului de operare preferată de utilizatorii avansați datorită ușurinței în utilizare (în special a ușurinței utilizării comenzilor).

În **TUI** interacțiunea cu sistemul de operare se realizează prin intermediul ferestrelor, meniurilor, butoanelor și mouse-ului. Pentru desenarea interfeței sunt utilizate caractere disponibile pe orice terminal text:

Figura 2-19 Text User Interface

Acest tip de interfață se apropie de modul în care arată o interfață grafică, însă datorită faptului că folosește caractere text, rezoluția ecranului este mai mică.

GUI Este un sistem de interacțiune cu utilizatorul de tipul WIMP (*window, icon, menu, pointing device*):

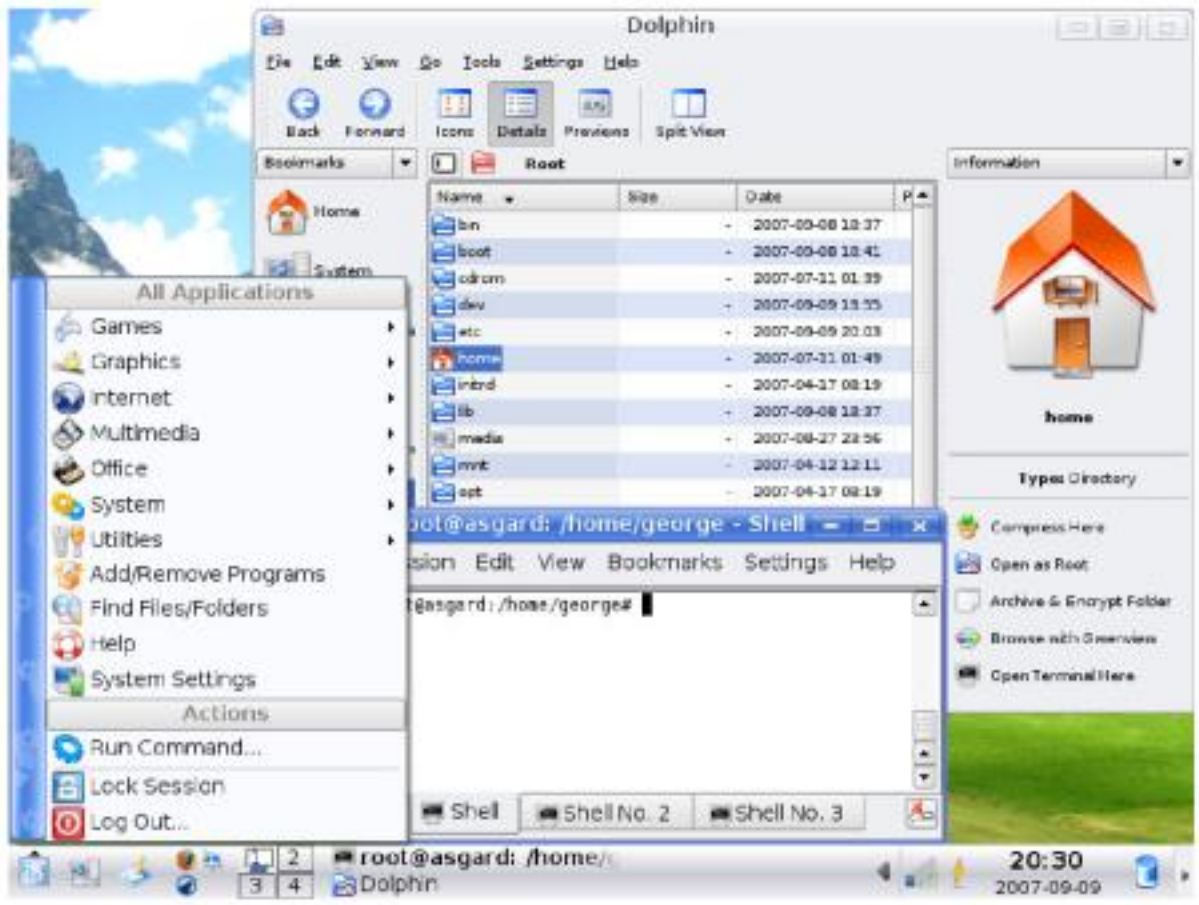


Figura 2-20 Graphical User Interface

Pentru desenarea interfeței GUI sunt folosite moduri grafice de rezoluție înaltă (VGA=640X480, SVGA=800X600, 1024X768, 1280X1024, 1600X1200, etc.)

2.4.2 Tipuri de console

O consolă este un tip de echipament care permite introducerea şi afişarea de text folosit în administrarea şi utilizarea sistemului de operare.

Inițial consolele erau formate dintr-un terminal (un monitor ce avea ataşată o tastatură) care se conecta la sistemul de calcul printr-o conexiune serială. Acest tip de console este foarte rar întâlnit în ziua de astăzi.

Pe un calculator obişnuit, monitorul împreună cu tastatură au acelaşi rol ca un terminal. Ele permit, în Linux, accesarea a două tipuri de console: terminalele virtuale (console în mod text) şi emulatoarele de terminale (console în mod grafic).

Terminalele virtuale au o rezoluție mică (în general 25 de linii cu 80 de caractere pe linie). Ele pot fi accesate, din interfață grafică, cu ajutorul tastelor CTRL+ALT+Fx, unde Fx reprezintă tastele F1 până la F6. Din aceste terminale virtuale revenirea în interfață grafică se face cu ajutorul tastelor (CTRL+)ALT+F7.

Trecerea dintr-un terminal virtual în altul poate realiza prin trei combinații de taste. Primele două sunt: **CTRL+ALT+Fx** şi **ALT+Fx** (unde x este numărul terminalului în care se va trece). A treia combinație de taste este **ALT+SageatăStânga** sau **ALT+SageatăDreapta** şi permite trecerea dintr-un terminal în cel precedent, respectiv în cel următor. De exemplu trecerea din terminalul 4 în terminalul 3 se poate realiza cu CTRL+ALT+F3 sau ALT+F3 sau ALT+SageatăStânga.

Valorile tastelor F prezentate sunt cele implicite pe majoritatea distribuțiilor. Aceste valori pot fi schimbate. De exemplu interfața grafică poate fi plasată pe terminalul trei sau pe terminalul cinci, sau numărul de terminale accesibile poate fi şase (valoarea implicită) sau zece.

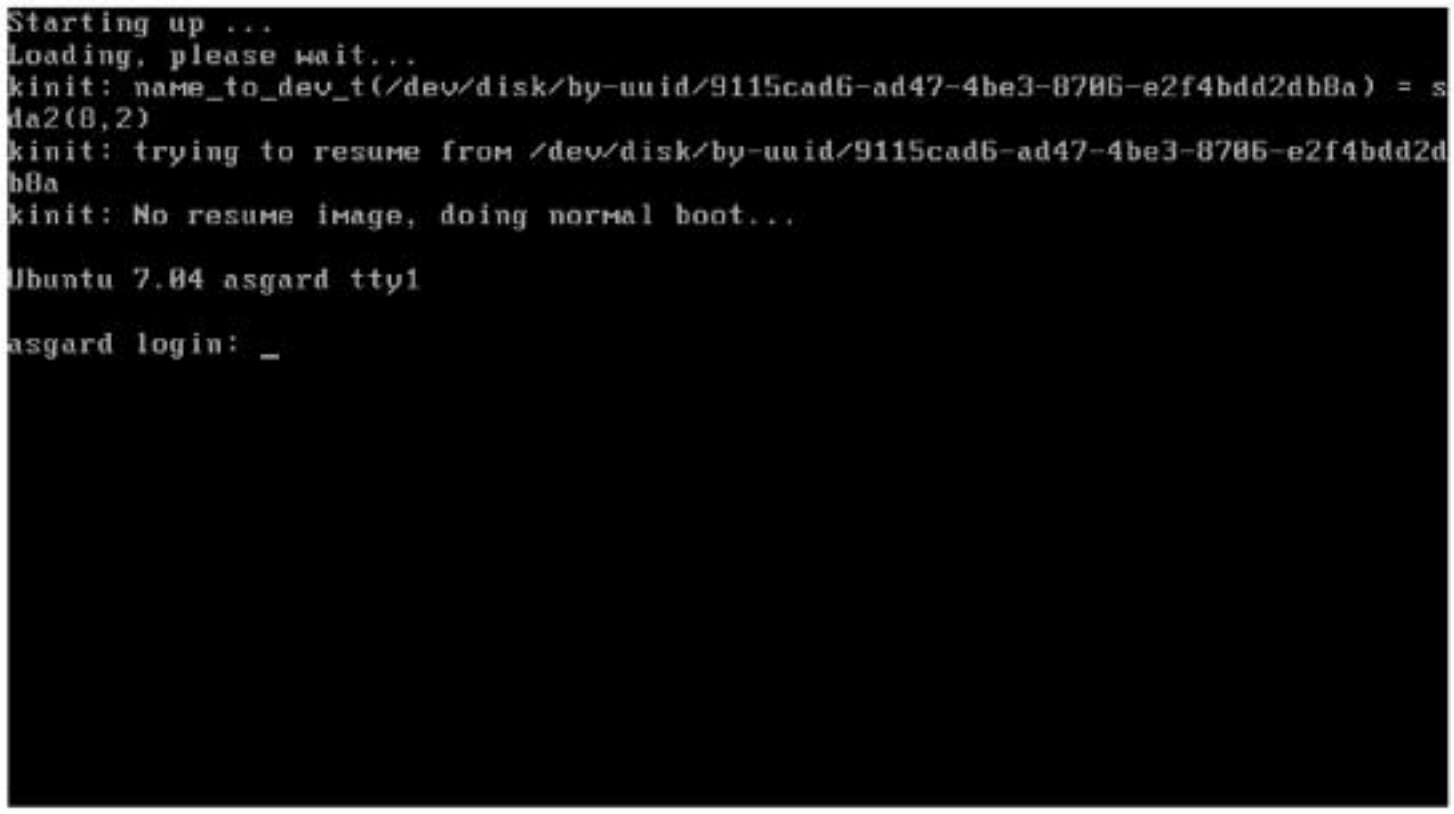


Figura 2-21 Un terminal virtual

Consolele în mod grafic reprezintă emulatoare de terminale. Dimensiunea lor pot fi extinse peste cea de 25x80 a terminalelor virtuale. De asemenea, un utilizator poate porni un număr nelimitat de astfel de emulatoare. Cele mai cunoscute emulatoare sunt Konsole (pentru KDE), gnome-terminal (pentru GNOME) şi xterm.

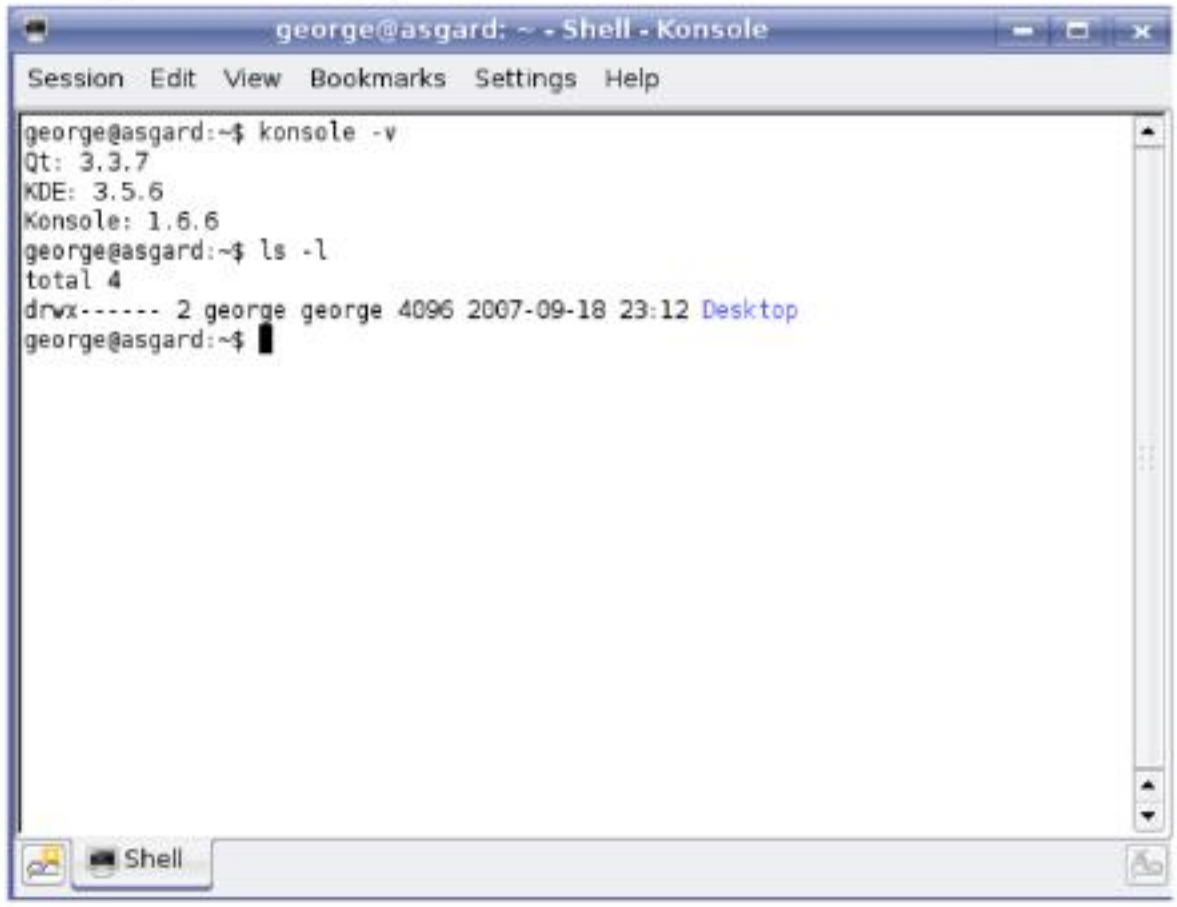


Figura 2-22 Konsole - un emulator de consolă

Atunci când un utilizator accesează un terminal virtual el trebuie să se autentifice folosind un nume de utilizator şi o parolă. La pornirea unui emulator de terminal utilizatorul nu mai trebuie să se autentifice, deoarece autentificarea s-a produs înainte de pornirea interfeţei grafice.

De asemenea, la pornirea unui emulator de terminal utilizatorul curent va fi cel care este autentificat în interfața grafică.

Pentru pornirea `Konsole` sau `gnome-terminal` se poate folosi meniul de start sau se poate folosi combinația de taste ALT+F2 care va porni o fereastra de *Run* (o fereastră în care se specifică numele unui program şi acesta va fi rulat). În această fereastră se va scrie numele Konsole sau gnome-teminal şi se va apăsa pe butonul *Run*.

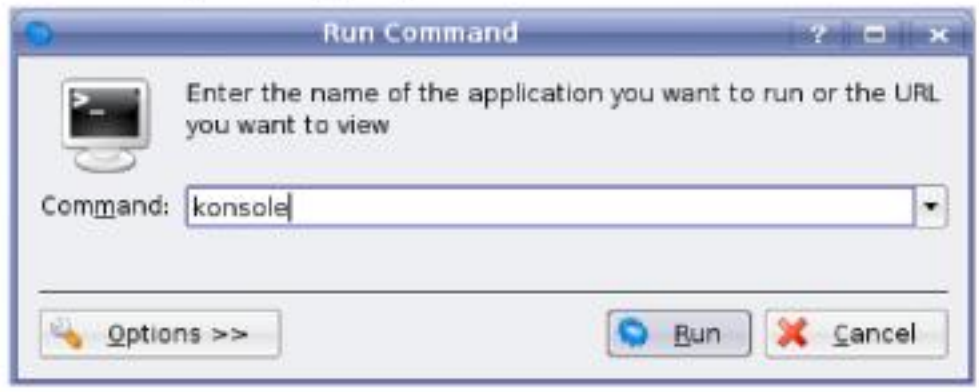


Figura 2-23 Pornirea Konsole

2.4.3 Shell

În IT denumirea de *shell* are mai multe sensuri. Unul dintre acestea este sensul de componentă software care realizează interfața cu utilizatorul. Această interfață poate fi una CLI, caz în care *shell*-ul este un interpretor de comenzi, sau una GUI, caz în care *shell*-ul este o interfață grafică dintre utilizator şi resursele sistemului de operare (cum ar fi *Explorer* în Windows).

Există mai multe aplicații shell de interpretare a comenzilor în Linux, cea mai uzuală fiind denumită bash (Bourne Again SHell).

2.4.4 Interfața în linie de comandă (CLI)

În ultimii ani, îndepărtarea de linia de comandă a sistemului de operare Linux este din ce în ce mai accentuată în detrimentul utilizării interfeței grafice, mai intuitive şi cu facilități mult mai avansate. Totuşi, interfața în linia de comandă rămâne o componentă esențială în configurarea rapidă a unui sistem şi a administrării de la distanță. Interfața în linia de comandă permite toate operațiile care pot fi realizate şi din interfața grafică.

2.4.4.1 Consola (prompt)

Comenzile Linux sunt transmise la consolă. Consola sau emulatorul de terminal rulează un interpretor de comenzi şi oferă utilizatorului un prompt. De obicei, acesta are un format de genul:

```
[username@localhost ~]$
```

La acest prompt, utilizatorul va putea introduce comenzi. **username** reprezintă numele utilizatorului, iar **localhost** este numele stației (*hostname*). Simbolul „~" va fi detaliat în continuare. Simbolul „$"semnifică terminarea prompt-ului şi începerea zonei de introducere a comenzilor.

De exemplu, în cazul următor:

```
george@asgard:~$
```

`george` este numele utilizatorului şi `asgard` este numele stației.

2.4.4.2 Comenzi

Comenzile sunt cuvinte cheie care se introduc la consolă (şi sunt interpretate de interpretorul de comenzi) şi sunt folosite pentru a configura sistemul de operare sau a obține informații de la acesta. Comenzile pot fi simple sau pot avea parametri. O **comandă simplă** este o comandă care conține un singur cuvânt cheie suficient pentru a executa o acțiune. Majoritatea comenzilor sunt însă **comenzi compuse** şi pot primi unul sau mai mulți parametri care vor afecta rezultatul comenzii. Separarea între numele comenzii şi parametru sau între doi parametri se realizează prin intermediul caracterului „spațiu". De multe ori primul parametru (sau şi alți parametri, funcție de caz) este marcat prin caracterul „-".

Mai jos este prezentat un exemplu de comandă simplă şi apoi aceeaşi comandă cu parametri. Comenzile au de obicei rezultate care sunt afişate pe consolă utilizatorului. În exemplul nostru, fiecare comandă are un rezultat care este afişat.

```
george@asgard:~$ ps
  PID TTY          TIME CMD
13391 pts/0    00:00:00 bash
13666 pts/0    00:00:00 ps

george@asgard:~$ ps -o uid,pid,cmd
  UID   PID CMD
 1000 13391 -bash
 1000 13669 ps -o uid,pid,cmd

george@asgard:~$
```

Setul cel mai important de comenzi este cel care interacționează cu sistemul de fişiere. Aceste comenzi sunt fundamentale pentru interacțiunea cu sistemul şi rularea altor comenzi importante.

2.4.4.3 Căutarea de ajutor

Daca se doreşte căutarea de informații în legătură cu o comandă (parametri, date de ieşire, funcționalitate, etc) distribuțiile de Linux pun la dispoziție mai multe opțiuni:

- Transmiterea parametrului --help unei comenzi pentru a afişa un sumar al parametrilor posibili ai acelei comenzi. De exemplu:

```
george@asgard:~$ cp --help
Usage: cp [OPTION]... [-T] SOURCE DEST
  or:  cp [OPTION]... SOURCE... DIRECTORY
  or:  cp [OPTION]... -t DIRECTORY SOURCE...
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

Mandatory arguments to long options are mandatory for short options too.
  -a, --archive                same as -dpR
      --backup[=CONTROL]       make a backup of each existing destination file
  -b                           like --backup but does not accept an argument
      --copy-contents          copy contents of special files when recursive
  -d                           same as --no-dereference --preserve=link
[...]
```

- Comanda `whatis` este folosită pentru a afişa o scurtă descriere a unei comenzi

```
george@asgard:~$ whatis ls
ls (1)               - list directory contents
george@asgard:~$ whatis pwd
pwd (1)              - print name of current/working directory
george@asgard:~$ whatis cp
cp (1)               - copy files and directories
george@asgard:~$ whatis poweroff
poweroff (8)         - stop the system.
```

- Comanda `apropos` afişează comenzile care au legătură cu cuvântul transmis ca parametru:

```
george@asgard:~$ apropos zip
bunzip2 (1)          - a block-sorting file compressor, v1.0.3
bzcmp (1)            - compare bzip2 compressed files
bzdiff (1)           - compare bzip2 compressed files
bzegrep (1)          - search possibly bzip2 compressed files for a regular expression
bzfgrep (1)          - search possibly bzip2 compressed files for a regular expression
[...]
```

- Comenzile `man` şi `info` sunt utilizate pentru a afişa paginile de documentaţie complete asociate unei comenzi dorite. Aceste pagini conţin informaţii detaliate despre opţiunile posibile ale comenzilor şi utilizările acestora. Comenzile `man` şi `info` primesc ca argument numele comenzii a cărei documentaţie se doreşte afişată:

```
george@asgard:~$ man ls
Reformatting ls(1), please wait...
george@asgard:~$ man pwd
Reformatting pwd(1), please wait...
george@asgard:~$ info coreutils ls
george@asgard:~$ info coreutils cp
```

2.4.4.4 Autentificarea şi părăsirea sistemului

Autentificarea locală în sistemul de operare se poate realiza folosind un terminal virtual sau se poate realiza în interfaţă grafică, printr-un *display manager* [vezi 12.1.2].

În cazul în care utilizatorul s-a autentificat în interfaţă grafică, el poate părăsi sistemul de operare (*logout*) tot din mediul grafic, folosind butonul de *logout*.

În cazul în care utilizatorul s-a autentificat într-un terminal virtual, el poate părăsi sistemul de operare folosind comanda `logout` sau comanda `exit`.

Comanda `logout` este folosită pentru a părăsi un *login shell*. Acesta este un shell creat atunci când utilizatorul s-a autentificat în sistem.

Comanda `exit` este folosită pentru a părăsi atât un login shell cât şi un shell creat din acest login shell (cu ajutorul comenzii `su` de exemplu).

2.4.5 Oprirea sistemului de calcul

Pentru a opri sau reporni un sistem de calcul se foloseşte comanda `shutdown`. Utilizarea acestei comenzi presupune ca utilizatorul să aibă drepturi administrative. Sintaxa comenzii este:

```
shutdown [OPTION]... TIME [MESSAGE]
```

Printre opţiunile folosite în mod uzual se numără: `-r` pentru a reporni staţia şi `-h` pentru a o opri. `time` reprezintă momentul la care va fi oprită sau repornită staţia, iar `message` este un mesaj ce va fi trimis pe toate terminalele legate la sistem.

De exemplu, pentru a opri staţia imediat, se rulează comanda:

```
root@asgard:/home/george# shutdown -h now

Broadcast message from george@asgard
        (/dev/pts/0) at 18:28 ...

The system is going down for halt NOW!
```

Pentru a reporni staţia imediat, se rulează comanda:

```
root@asgard:/home/george# shutdown -r now
Broadcast message from george@asgard
        (/dev/pts/4) at 11:04 ...

The system is going down for reboot NOW!
```

Pentru a opri staţia la ora 20:00:

```
root@asgard:/home/george# shutdown -r 20:00

Broadcast message from george@asgard
        (/dev/pts/4) at 11:03 ...

The system is going down for reboot in 537 minutes!
```

Pentru a opri stația în 10 minute de la executarea comenzii se foloseşte comanda:

```
root@asgard:/home/george# shutdown -r +10

Broadcast message from george@asgard
        (/dev/pts/0) at 18:33 ...

The system is going down for reboot in 10 minutes!
```

O altă posibilitate de a reporni stația este utilizarea comenzii `restart`. Pentru oprirea stației se pot folosi şi comenzile `halt` sau `poweroff`. Toate aceste comenzi cer ca utilizatorul care le execută să fie root.

2.4.6 Obținerea de drepturi administrative în sistem: sudo, su

În sistemele de operare Windows, utilizatorul normal are de cele mai multe ori şi drepturi de administrare şi acest lucru poate crea probleme de securitate. [vezi 9.2]. În sistemele de operare Linux se face o distincție clară între drepturile de administrare ale sistemului şi drepturile de utilizare ale sistemului. De aceea un singur utilizator are drepturi de administrare depline. Acest utilizator este unul special şi se numeşte **root**.

Root are drepturi depline în sistemul de operare. El poate realiza configurări, poate modifica modul în care porneşte sistemul de operare, poate acorda drepturi parțiale altor utilizatori, etc.

Pentru a putea realiza configurări, acțiuni pe care doar utilizatorul root le poate realiza, un utilizator are trei posibilități.

Prima posibilitate este de se autentifica în sistemul de operare folosind numele de utilizator root şi parola asociată acestuia. În cele mai multe sisteme de operare folosite în producție autentificarea ca root este dezactivată din motive de siguranță. De aceea această metoda poate sa nu funcționeze.

A doua posibilitate este ca utilizatorul să se autentifice în sistem cu numele sau de utilizator şi cu parola asociată, creând astfel un *login shell*. După ce s-a autentificat el poate să pornească un nou *shell* în care să se autentifice ca un alt utilizator. Atunci când părăseşte *shell*-ul nou creat (de exemplu prin comanda `exit`) el va reveni în *shell*-ul anterior (cel de *login*). Pornirea unui nou *shell* în care se realizează autentificarea ca alt utilizator se face cu ajutorul comenzii `su` (*switch user*). Această comandă poate primi ca parametru numele utilizatorului în contul căruia se va face autentificarea:

```
george@asgard:~$ su uso
Password:
uso@asgard:/home/george$ exit
exit
george@asgard:~$
```

Din motive de securitate caracterele tastate în dreptul parolei nu vor fi afişate.

În cazul în care comanda `su` nu primeşte nici un parametru, va fi pornit un *shell* în care se va încerca autentifica utilizatorul root.

Pentru a părăsi un *shell* creat cu `su` nu se poate folosi comanda `logout`, deoarece sesiunea autentificată în sistem (*login shell*) aparține tot utilizatorului inițial (utilizatorul inițial fiind

`george` în exemplul de mai jos) şi nu noului utilizator (noul utilizator fiind `uso` în exemplul de mai jos):

```
george@asgard:~$ su uso
Password:
uso@asgard:/home/george$ logout
bash: logout: not login shell: use `exit'
uso@asgard:/home/george$ exit
exit
george@asgard:~$
```

Ubuntu şi Kubuntu au dezactivat în mod implicit contul de root, nefiind configurată nici o parolă pentru acesta. De aceea, autentificarea ca root, fie că este încercată la etapa de autentificare în sistem, fie că este realizată cu `su`, va eşua.

În acest caz se poate încerca **a treia posibilitate** de obținere a drepturilor administrative: utilizarea comenzii `sudo`. Această comandă permite executarea unui program ca alt utilizator. Spre deosebire de `su`, care cerea parola utilizatorului al cărui cont va fi utilizat, `sudo` cere parola utilizatorului curent. Din acest motiv comanda `sudo su` va avea succes (este cerută parola utilizatorului curent, nu cea a utilizatorului root, după care comanda `su` este executata ca root):

```
george@asgard:~$ sudo su
Password:
root@asgard:/home/george#
```

Atunci când utilizatorul root foloseşte comanda `sudo` pentru a porni un shell în care să se autentifice ca alt utilizator, nu i se va cere nici o parolă.

2.5 Configurări de bază ale SO

2.5.1 Asigurarea conectivității la Internet

Una dintre primele configurări ce trebuie realizată pe sistemul de operare instalat este asigurarea conectivității la Internet. Kubuntu este o distribuție de Linux ce se bazează pe pachete şi necesită o conexiune la Internet pentru a putea copia şi instala versiuni mai recente ale pachetelor, realizând în acest fel actualizarea întregului sistem de operare.

Pentru asigurarea conectivității la Internet există mai multe alternative. Cea mai simplă variantă este aceea în care toți parametrii de rețea [vezi 7.2] (adresă IP, mască de rețea, gateway, server DNS) sunt configurați în mod automat folosind DHCP. Această metodă este cea încercată implicit de sistemul de operare. Dacă aceasta reuşeşte, utilizatorul are acces la Internet, nemaifiind necesară o altă configurare.

Dacă încercarea de configurare folosind DHCP eşuează, utilizatorul va trebui să configureze manual parametrii de rețea. Acest lucru se poate realiza temporar (efectele configurărilor vor dura până la următoarea pornire a sistemului de operare sau până la o nouă configurare) sau permanent (efectele configurărilor se vor păstra şi după repornirea sistemului de operare).

Configurările temporare pentru adresa IP, masca de rețea şi gateway se realizează folosind comenzile `ifconfig` şi `route`. Pentru a putea folosi aceste comenzi utilizatorul trebuie să obțină drept de administrare.

2.5.2 Administrarea utilizatorilor şi a grupurilor de utilizatori

UID, GID

Fiecărui utilizator din sistem îi este asociat un identificator numit UID (*User Identifier*). UID este un număr întreg ce poate lua valori între 0 şi 32767 sau 65535, funcţie de distribuţie. Valoarea 0 este rezervată pentru utilizatorul root.

Pentru a găsi identitatea unui utilizator sistemul de operare foloseşte UID, şi nu numele utilizatorului. De aceea, dacă în fişierul `/etc/passwd` se înlocuieşte UID-ul unui utilizator normal cu 0, acel utilizator va căpăta drepturi administrative egale cu ale utilizatorului root.

Pe sistemele de operare Linux utilizatorii pot fi organizaţi în grupuri. În acest fel se pot realiza diverse configurări administrative mai uşor, aplicându-se politici la nivelul întregului grup şi nu per utilizator. Un grup poate conţine mai mulţi utilizatori. Un utilizator poate face parte din mai multe grupuri.

Pentru fiecare grup sistemul de operare asociază un identificator, numit GID (*Group Identifier*). Acest identificator este un număr întreg ce poate lua valori între 0 şi 32767 (limita superioară poate să difere în funcţie de distribuţie). GID 0 este rezervat pentru grupul utilizatorului root. La fel ca la UID, sistemul de operare foloseşte GID pentru a identifica un anumit grup şi nu numele grupului.

Adăugarea şi ştergerea utilizatorilor

Adăugarea unui utilizator poate fi realizată doar de root, în două moduri. Primul dintre ele este utilizarea comenzii `useradd`. Utilizarea acestei comenzi este greoaie, motiv pentru care adăugarea unui utilizator se realizează în mod uzual cu ajutorul comenzii `adduser`. `adduser` este un script [vezi 11] care apelează la rândul lui `useradd`.

Apelarea `adduser` se poate realiza cu următorii parametri:

```
  adduser [options] [--home DIR] [--shell|-s SHELL] [--no-create-home]
[--uid ID] [--firstuid ID] [--lastuid ID] [--ingroup GROUP | --gid ID] [--
disabled-password] [--disabled-login] [--gecos GECOS] [--add_extra_groups]
user
```

Dacă `adduser` este apelat doar cu numele utilizatorului ce va fi creat, el va porni un asistent care va cere toate datele necesare creării contului:

```
root@asgard:/home/george# adduser uso
Adding user `uso' ...
Adding new group `uso' (1001) ...
Adding new user `uso' (1001) with group `uso' ...
Creating home directory `/home/uso' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for uso
Enter the new value, or press ENTER for the default
        Full Name []: uso
        Room Number []: EG306
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [y/N] y
root@asgard:/home/george#
```

Din motive de securitate caracterele tastate în dreptul parolei nu vor fi afişate.

Odată cu crearea unui utilizator, este creat şi un grup cu acelaşi nume, la care este adăugat utilizatorul creat.

Ştergerea unui utilizator se poate realiza cu `userdel` sau cu `deluser` (`deluser` este un script care apelează la rândul lui `userdel`):

```
root@asgard:/home/george# deluser uso
Removing user `uso' ...
Done.
```

Adăugarea şi ştergerea unui grup de utilizatori

Adăugarea unui grup de utilizatori se poate realiza cu ajutorul comenzii `groupadd`, sau cu ajutorul comenzii `addgroup`. `addgroup` este un script care apelează la rândul lui `groupadd`.

```
root@asgard:/home/george# addgroup usogroup
Adding group `usogroup' (GID 1002) ...
Done.
```

Ştergerea unui grup de utilizatori se realizează tot prin două comenzi, `groupdel` şi `delgroup` (`delgroup` este un script care apelează `groupdel`).

```
root@asgard:/home/george# delgroup usogroup
Removing group `usogroup' ...
Done.
```

Modificarea datelor unui utilizator

Modificarea datelor unui utilizator se realizează cu ajutorul comenzii `usermod`. Aceasta permite modificarea grupului din care face parte un utilizator, a directorului de bază (*home*), a numelui de utilizator, a parolei, etc.

De exemplu, pentru modifica directorul de bază al utilizatorului `george` se foloseşte opţiunea `-d`:

```
uso@asgard:~$ cd ~
uso@asgard:~$ pwd
/home/ uso
uso@asgard:~$ sudo su
root@asgard:/home/uso# usermod -d /home/usodir/ uso
root@asgard:/home/uso# exit
exit
uso@asgard:~$ exit
uso@asgard:~$ pwd
/home/uso
uso@asgard:~$
```

În exemplul anterior a fost schimbat directorul curent în directorul de bază al utilizatorului `uso`, apoi a fost afişat directorul curent (pentru a vedea care este directorul de bază actual), a fost pornit un nou *shell* în care utilizatorul autentificat a fost root, a fost modificat directorul de bază al utilizatorului `uso` în `/home/usodir`, a fost închis shell-ul de root, apoi şi utilizatorul uso a părăsit sistemul de operare. Schimbarea directorului de bază al unui utilizator se realizează doar după ce utilizatorul părăseşte sistemul şi se autentifică din nou:

```
uso@asgard:~$ pwd
/home/usodir/
```

Modificarea parolei unui utilizator se poate face şi cu ajutorul comenzii `passwd`:

```
root@asgard:/home/uso# passwd uso
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Din motive de securitate caracterele tastate în dreptul parolei nu vor fi afişate.

Dacă nu este specificat nici un nume de utilizator va fi schimbată parola utilizatorului curent. Doar utilizatorul root poate schimba parola altui utilizator.

2.6 Studii de caz

2.6.1 Interoperabilitatea sistemului de fişiere între Linux şi Windows

Atunci când pe nu calculator este instalat şi un sistem de operare Windows şi unul Linux se pune problema partajării de fişiere între cele două. Ca exemplu concret, cum se pot accesa din Linux (pentru scriere şi citire) datele aflate în Windows, şi cum se pot accesa din Windows datele aflate în Linux.

Soluţia la această problemă se bazează pe ideea că pentru datele aflate pe cel puţin o partiţie se va asigura suportul de citire şi de scriere atât din Linux cât şi din Windows. În acest sens există trei variante.

Prima dintre ele constă în utilizarea unei partiţii FAT32 pentru a stoca datele ce trebuie partajate. Avantajul acestei soluţii este faptul că atât în Windows cât şi în Linux există suport nativ pentru sistemul de fişiere FAT. Dezavantajele majore sunt limitarea dimensiunii unei partiţii FAT32 la maxim 32GB şi limitarea dimensiunii maxime a unui fişier la 4GB (ca atare pe o astfel de partiţie nu se poate salva imaginea unui disc DVD de exemplu).

A doua soluţie o reprezintă utilizarea unei partiţii ext2 sau ext3 pentru stocarea datelor partajate. Această soluţie nu prezintă limitările partiţiei FAT, însă Windows-ul nu suportă nativ acest tip de partiţii. Pentru a putea accesa din Windows o partiţie est2 sau ext3 trebuie instalat un driver special. Acest driver se numeşte „ext2 IFS" şi este disponibil la adresa http://www.fs-driver.org/.

Acest driver oferă suport de scriere şi citire pentru sistemul de fişiere ext2. Sistemul de fişiere ext3 este asemănător cu ext2, diferenţa dintre ele fiind reprezentată de suportul de jurnalizare pe care îl oferă ext3. „ext2 IFS" poate citi şi scrie date de pe o partiţie ext3, însă nu va folosi suportul de jurnalizare pe care îl oferă aceasta.

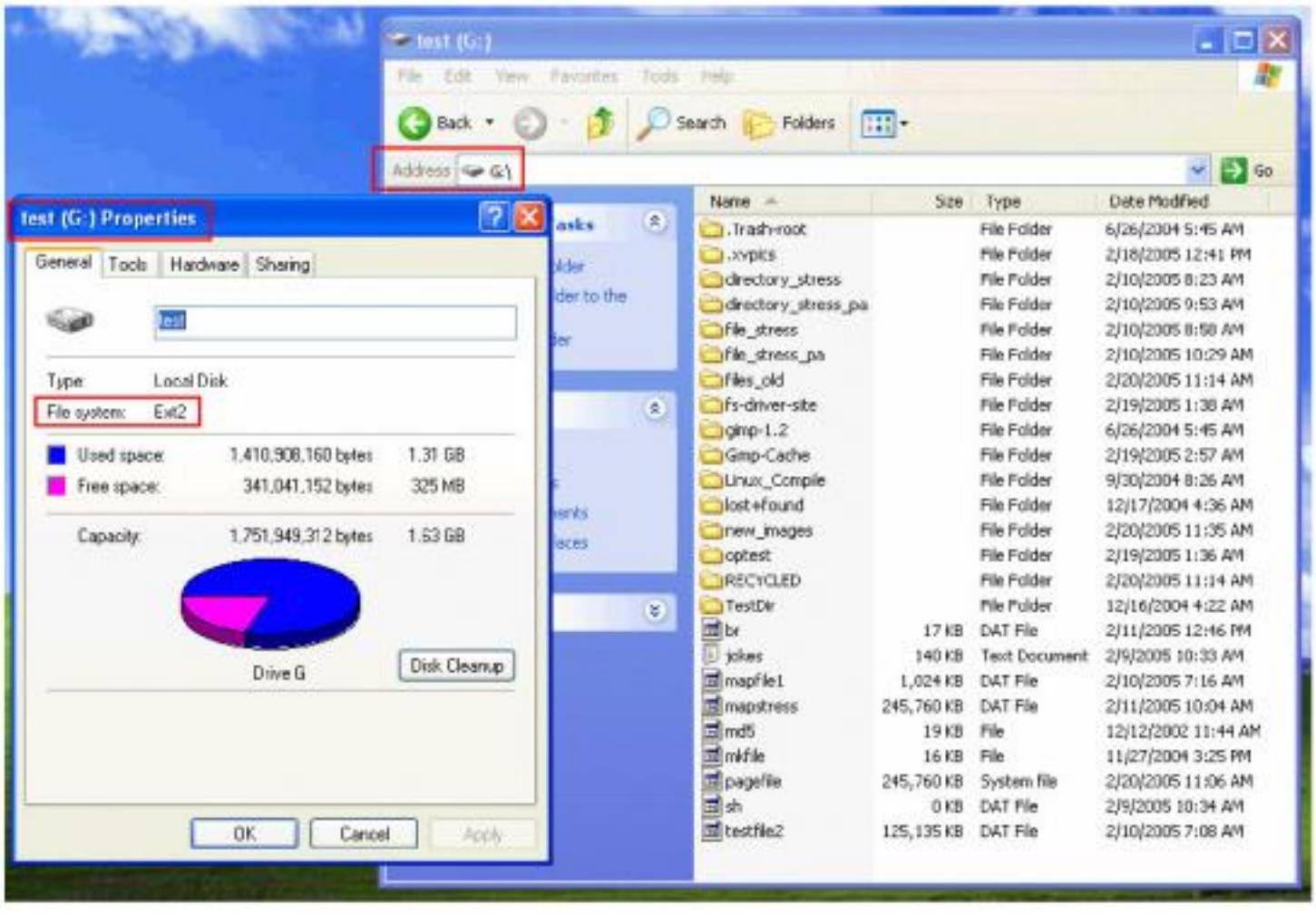


Figura 2-24 Accesarea unei partiţii ext2 folosind ext2fs

A treia variantă o reprezintă accesarea unei partiții NTFS din Linux. În Linux există suport pentru citirea datelor de pe un sistem de fişiere NTFS. Datorită faptului că standardul NTFS este unul închis, dezvoltarea unui driver care să ofere suport de scriere a fost greu de realizat. În momentul de față există însă un driver care oferă acest suport. El se numeşte ntfs-3g si poate fi găsit la adresa http://www.ntfs-3g.org/.

2.6.2 GParted

GParted este o aplicație de partiționare specifică GNOME. Ea poate crea, şterge, redimensiona, muta, verifica şi copia partiții. Sistemele de fişiere suportate de GParted sunt următoarele:

	Detectare	Citire	Creare	Mărire	Micşorare	Mutare	Copiere	Verificare
ext2	Da	Da	Da	Da	Da	Da	Da	Da
ext3	Da	Da	Da	Da	Da	Da	Da	Da
Extended	Da	Da	Da	Parțial	Parțial	Nu	Nu	Nu
FAT16	Da	Da	Da	Da	Da	Da	Da	Da
FAT32	Da	Da	Da	Da	Da	Da	Da	Da
HFS	Da	Da	Da	Nu	Da	Da	Da	Nu
HFS+	Da	Da	Nu	Nu	Da	Da	Da	Nu
JFS	Da	Da	Da	Da	Nu	Da	Da	Da
Swap	Da	Nu	Da	Da	Da	Da	Da	Nu
NTFS	Da	Da	Da	Da	Da	Da	Da	Da
Reiser4	Da	Da	Da	Nu	Nu	Da	Da	Da
ReiserFS	Da	Da	Da	Da	Da	Da	Da	Da
UFS	Da	Nu	Nu	Nu	Nu	Da	Da	Nu
XFS	Da	Da	Da	Da	Parțial	Da	Da	Da

Figura 2-25 Sistemele de fițiere suportate de GParted

Detectarea majorității partițiilor se realizează cu ajutorul bibliotecii GNU libparted. În cazul în care această bibliotecă nu poate detecta o anumită partiție se folosesc o serie de unelte separate, a căror necesitate şi încărcare este detectată în momentul rulării (*runtime*).

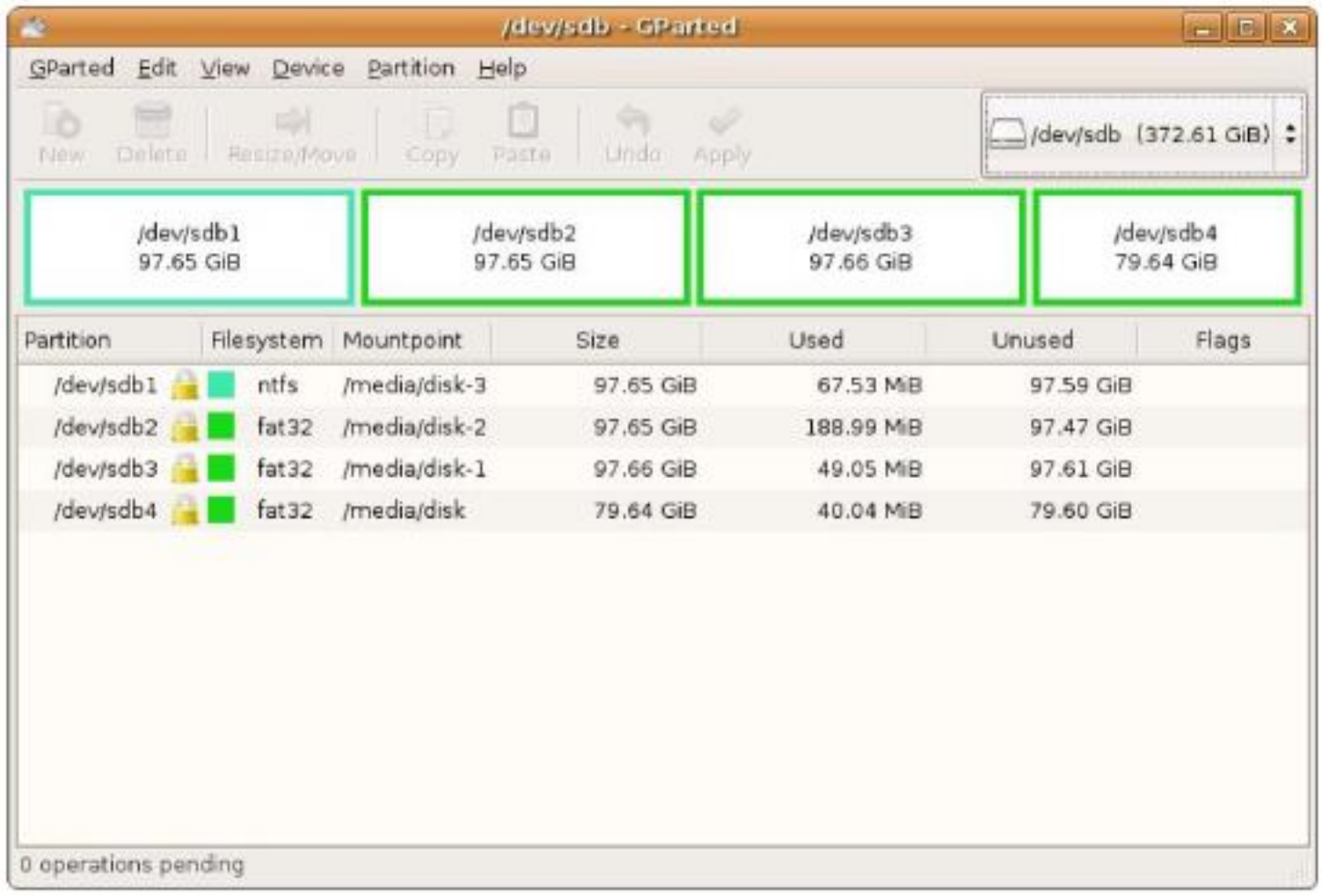


Figura 2-26 GParted

GParted oferă şi o versiune disponibilă sub forma unui LiveCD. Aceasta este bazată pe Gentoo şi permite administrarea partiţiilor în cazul în care sistemul de operare nu mai porneşte. De asemenea permite realizarea unei copii de siguranţă a datelor în aceleaşi condiţii.

2.6.3 Fişierele în care sunt stocate informaţii despre utilizatori

Informaţiile despre grupuri şi utilizatori sunt stocate în trei fişiere text:

- `/etc/passwd` – informaţii despre conturile utilizatorilor
- `/etc/shadow` – informaţii despre parolele utilizatorilor
- `/etc/group` – informaţii despre grupurile de utilizatori

Fişierul `/etc/passwd` conţine câte o linie pentru fiecare utilizator. Această linie are şapte câmpuri despărţite prin caracterul „:”. Câmpurile sunt (în ordinea în care apar în fişier):

- Nume de utilizator
- Parola. Un caracter `x` în această poziţie arată faptul că parola este stocată în fişierul `/etc/shadow`.
- UID
- GID
- Informaţii legate de utilizator (Nume, Adresa, Telefon)
- Directorul de bază
- Shell-ul folosit

Un exemplu de fişier `/etc/passwd` este:

```
root@asgard:/home/george# cat /etc/passwd
[...]
sync:x:4:65534:sync:/bin:/bin/sync
george:x:1000:1000:George,,,:/home/george:/bin/bash
uso:x:1001:1001:uso,EG306,,:/home/uso:/bin/bash
[...]
```

Fişierul `/etc/shadow` conţine şi el câte o linie pentru fiecare utilizator. Pe fiecare linie se găsesc opt câmpuri despărţite prin caracterul „:". Câmpurile sunt (în ordinea în care apar în fişier):

- Numele de utilizator
- Parola în format criptat. Dacă în loc de parolă se află un caracter „!", utilizatorul respectiv nu se poate autentifica în sistem
- Data la care a fost realizată ultima schimbare de parolă (data este notată în numărul de zile trecute de la 1 ianuarie 1970)
- Numărul minim de zile care pot trece între două schimbări de parolă
- Numărul maxim de zile pentru care parola este activă
- Cu câte zile înainte de expirarea parolei este anunţat utilizatorul să îşi shimbe parola
- Numărul de zile care trec după expirarea parolei până când contul va fi dezactivat
- Data la care contul va expira (data este notată în numărul de zile trecute de la 1 ianuarie 1970)

Un exemplu de fişier `/etc/shadow` este:

```
root@asgard:/home/georgedir/# cat /etc/shadow
root:!:13774:0:99999:7:::
daemon:*:13620:0:99999:7:::
bin:*:13620:0:99999:7:::
sys:*:13620:0:99999:7:::
sync:*:13620:0:99999:7:::
george:$1$LBNk03w4$4O2R/uMfCs/R9LOmH6zDG0:13774:0:99999:7:::
uso:$1$IPSj/ETo$1Uqe1Nrqm0r4LYdMihh6x1:13775:0:99999:7:::
[...]
```

Fişierul `/etc/group` conţine câte o linie pentru fiecare grup. Această linie are patru câmpuri despărţite prin caracterul „:". Câmpurile sunt (în ordinea în care apar în fişier):

- Numele grupului
- Parola (câmp ce nu mai este folosit)
- GID
- Numele utilizatorilor ce fac parte din grup, în afară de utilizatorul care are acelaşi nume cu numele grupului

Un exemplu de fişier `/etc/group` este:

```
root@asgard:/home/georgedir/# cat /etc/group
root:x:0:
lp:x:7:cupsys
scanner:x:104:cupsys,hplip,george
george:x:1000:
uso:x:1001:
[...]
```

Cuvinte cheie

- CLI
- /etc/passwd
- consolă
- distribuție Linux
- gestionare utilizatori
- GUI
- Kubuntu
- LiveCD
- MBR
- mediu grafic
- partiție
- shell
- su
- sudo
- TUI
- UID

Întrebări

Câți utilizatori cu identificatorul 0 pot exista pe un sistem cu o partiție primară și două extinse:

- ❑ nici unul
- ❑ 1
- ❑ 2
- ❑ oricâți

Care dintre următoarele nu este necesară pentru a rula un sistem de operare de pe LiveCD:

- ❑ configurarea unei partiții primare
- ❑ BIOS
- ❑ RAM
- ❑ procesor

Din ce sisteme de operare nu se pot realiza configurării din TUI:

- ❑ Windows Vista
- ❑ Debian
- ❑ ubuntu LiveCD
- ❑ PalmOS

Operația de repornire a sistemului poate fi inițiată doar de utilizatorul privilegiat. În urma repornirii sistemului tot conținutul memoriei RAM este golit.

- ❑ adevărat, adevărat
- ❑ adevărat, fals
- ❑ fals, adevărat
- ❑ fals, fals

Care dintre variante de adevărată, dacă la execuția comenzii de mai jos se obține un prompt de shell fără a se solicita autentificare:

```
sudo rrazvan
```

- ❑ comanda a fost lansată de utilizatorul root
- ❑ comanda a fost lansată de utilizatorul rrazvan
- ❑ comanda a fost lansată din directorul /etc
- ❑ comanda a fost lansată din directorul /root

3 Sistemul de fişiere

Nine megs for the secretaries fair,
Seven megs for the hackers scarce,
Five megs for the grads in smoky lairs,
Three megs for system source;

One disk to rule them all,
One disk to bind them,
One disk to hold the files
And in the darkness grind 'em.

Ce se învaţă din acest capitol?

- Ce este şi cum se utilizează sistemele de fişiere
- Tipuri de sisteme de fişiere
- Utilizarea căilor relative şi absolute
- Lucrul cu fişiere (creare, copiere, mutare etc.)
- Descriptori de fişiere
- Redirectarea comenzilor în fişiere
- Arhivarea şi dezarhivarea
- Drepturi de acces la fişiere

3.1 Noţiuni introductive

Fişierul (*file*) este reprezentarea logică a unei informaţii sub forma unei înşiruiri de octeţi. Fişierul poate fi considerat ca fiind versiunea modernă a documentului scris. Directorul (*directory*) este o entitate în care se pot regăsi fişiere şi/sau alte directoare. Acesta poate fi considerat versiunea modernă a dosarului. În interfeţele grafice, directorul este de obicei denumit *folder*.

Sistemele de operare folosesc fişierele pentru a organiza date, indiferent dacă aceste date sunt ale utilizatorului sau sunt generate pe moment de sistem.

Spre exemplu, în Linux şi în Mac OS, orice informaţie se găseşte într-un fişier:

- datele se regăsesc în fişiere
- directoarele sunt şi ele fişiere, dar cu atributul de director
- fiecare dispozitiv poate fi accesat printr-un fişier, inclusiv mouse-ul, memoria şi placa video

Intern, Windows-ul foloseşte o schemă asemănătoare structurii de fişiere şi directoare pentru a denumi dispozitivele, dar această structură nu se suprapune peste structura de fişiere precum în Linux şi în Mac OS.

3.1.1 Ce este un sistem de fişiere

3.1.1.1 Noţiuni de bază

Fişierele organizează informaţiile pe mediile de stocare. Mediile de stocare pot fi considerate spaţii continue de octeţi. În aceste spaţii, se pot regăsi mai multe fişiere, de dimensiuni variabile.

Sistemul de fişiere reprezintă modul de organizare a fişierelor într-un mediu de stocare pentru a le face mai uşor accesibile. Organizarea include atât partea logică (modul în care sunt adresate fişierele) cât şi partea fizică (modul în care sunt stocate fişierele ca înşiruire de octeţi).

Orice mediu de stocare poate fi utilizat pentru scrierea şi citirea de fişiere dacă se cunoaşte sistemul de fişiere utilizat pentru organizarea mediului.

În timp au apărut mai multe tipuri de sisteme de fişiere. În mediul open-source s-au dezvoltat şi încă se dezvoltă cele mai multe dintre ele. Câteva dintre cele mai importante sisteme de fişiere utilizate în prezent sunt menţionate în tabelul următor, împreună cu sistemele de operare în care se întâlnesc.

SO Sistem fişiere	Windows	Linux	Mac OS
FAT32	Nativ	Nativ	Nativ
NTFS	Nativ (dupa WinNT)	prin ntfs-3g[1]	prin ntfs-3g
Ext2/Ext3	Driver third-party[2]	Nativ	-
ReiserFS	Aplicaţie third-party[3]	Nativ	-
HFS+	-	Nativ	Nativ
ISO9660	Nativ	Nativ	Nativ
UDF	Nativ	Nativ	Nativ

Figura 3-1 Sisteme de operare şi sisteme de fişiere suportate

Prin „Nativ" se înţelege că suportul este oferit prin driver-e ce însoţesc sistemul de operare (în cazul Linux însoţesc kernel-ul).

3.1.1.2 Tipuri de sisteme de fişiere

Sistemele de fişiere se pot clasifica şi după locul unde se găsesc datele. Tabelul următor prezintă succint această clasificare.

Tip	Exemplu	Descriere
sisteme de fişiere cu suport fizic	FAT32, NTFS, Ext3, ReiserFS, HFS+	se regăsesc de obicei pe un mediu de stocare
sisteme de fişiere virtuale	procfs, devfs SSHFS	conţin fişiere/date generate de SO (informaţii despre sistem) sau de o altă componentă software (alte surse)
sisteme de fişiere pentru reţea	NFS, SMB	utilizate pentru accesul la fişiere aflate în reţea

Figura 3-2 Clasificarea sistemeleor de fişiere după suportul datelor

---

[1] *ntfs-3g - http://www.ntfs-3g.org/*
[2] *Ext2 IFS for Windows - http://www.fs-driver.org/*
[3] *YAReG – Yet Another R(eiser)FStool GUI - http://yareg.akucom.de/*

Tabelul următor prezintă câteva sisteme de fişiere mai speciale şi aplicaţiile lor.

Tip sistem de fişiere	Sisteme de operare	Descriere
procfs	Linux (parţial suport în Mac OS prin FUSE)	reţine informaţii despre starea sistemului în prezent, precum date despre procesele din sistem, memorie etc. [vezi capitolul 6]
devfs	Linux/Mac OS	conţine câte un fişier pentru fiecare dispozitiv prezent în computer – citirile şi scrierile din fişiere sunt mediate de driver-ele dispozitivelor
ramfs	Linux/Mac OS	Utilizat pentru a încărca la boot-are fişiere direct în RAM, pentru un acces mai rapid
GFS	utilizat intern de Google	Google File System[4] – sistem de fişiere utilizat intern de Google în cluster-e distribuite
UnionFS	Linux	UnionFS[5] permite accesul transparent la un sistem de fişiere realizat prin suprapunerea directoarelor din diferite sisteme de fişiere.
JFFS2	Linux	Journaling Flash File System 2 – JFFS2[6] - este un sistem de fişiere optimizat pentru a minimiza numărul de scrieri în memoriile flash folosite în USB stick-uri (memoriile flash au un număr limitat de cicli de scriere) şi a folosi celulele de memorie cât mai uniform. Acest sistem de fişiere este folosit pe în proiectul OLPC[7].

Figura 3-3 Sisteme de fişiere speciale

Explicaţie UnionFS (folosind noţiuni prezentate în continuare în acest capitol): presupunând că directorul radacină „/" este montat de pe un CD-ROM în mod read-only, se poate monta „peste" acesta un sistem de fişiere read-write (în `/home` spre exemplu) de pe un dispozitiv care suportă scrierea (ex: un USB stick) – se obţine astfel un sistem de fişiere complet, cu directorul `/home` read-write.

3.1.1.3 Integritatea datelor

Un sistem de fişiere trebuie să asigure integritatea datelor chiar şi în cazul unei căderi (*failure*). De cele mai multe ori, această funcţionalitate este implementată prin jurnalizare.

> ***Jurnalizarea** este activitatea prin care se realizează un jurnal cu toate modificările efectuate asupra unui sistem de fişiere. Fiecare operaţie care se execută asupra sistemului de fişiere este întâi scrisă în jurnal. Operaţia poate presupune modificări în mai multe locuri din sistemul de fişiere.*

Driver-ul sistemului de fişiere execută la un anumit interval de timp operaţiile din jurnal pentru a actualiza starea discului. O operaţie se şterge din jurnal doar după ce toate modificările asociate ei au fost executate. În acest fel, în cazul apariţiei unei căderi (întrerupere de curent etc.), se pot executa operaţiile din jurnal care au rămas neexecutate (operaţiile începute şi care au fost întrerupte sunt încă prezente în jurnal şi vor fi reexecutate în întregime).

---

[4] *Google File System (GFS) - http://en.wikipedia.org/wiki/Google_File_System*
[5] *UnionFS - http://www.am-utils.org/project-unionfs.html*
[6] *Journaling Flash File System (JFFS2) - http://sourceware.org/jffs2/*
[7] *OLPC – One Laptop Per Child – http://laptop.org*

3.1.2 Alegerea unui sistem de fişiere

În momentul în care trebuie ales un sistem de fişiere, cele mai utilizate criterii de selecţie sunt:

- disponibilitatea (în primul rând) – dacă se poate folosi sistemul respectiv de fişiere în sistemul de operare ales (sau în mai multe sisteme de operare)
- gradul de siguranţă – majoritatea sistemelor de fişiere existente în prezent au jurnalizare
- restricţii speciale date de modul de organizare a datelor în sistemul de fişiere – exemplu: la FAT32 se folosesc doar 32 biţi pentru a stoca dimensiunea unui fişier, deci dimensiunea maximă a unui fişier este 4GB (mai puţin decât o imagine de DVD, fapt ce îl face nepractic pentru o parte din aplicaţiile multimedia)
- optimizări de performanţă, realizate spre exemplu în funcţie de tipul suportului de stocare, în funcţie de dimensiunea fişierelor etc.

Tabelul următor realizează o analiză sumară a caracteristicilor sistemelor de fişiere.

Tip sistem de fişiere	Sisteme de operare	Dimensiune maximă fişier	Jurna-lizare	Observaţii
FAT32	Windows/ Linux/ Mac OS	4 GB	nu	Cel mai folosit sistem de fişiere – întâlnit în mod special pe USB stick-uri, fără drepturi de acces; Windows-ul limitează la creare dimensiunea unei partiţii la 32GB, dar poate citi partiţii mai mari realizate şi formatate cu aplicaţii third-party folosit pentru a asigura compatibilitatea cu dispozitive sau sisteme mai vechi
NTFS	Windows/ Linux/ Mac OS	16TiB	da	Singurul sistem de fişiere pentru Windows recomandat de Microsoft. Singurul sistem de fişiere pentru Windows cu securitate.
Ext2/Ext3	Linux/ Windows[8]	16GiB - 64TiB	ext2-nu ext3-da	Sistemul de fişiere considerat cel mai stabil datorită unei istorii de dezvoltare foarte lungi; ext3 e compatibil cu versiunea anterioară ext2, aduce jurnalizare faţă de ext2
ReiserFS	Linux	8 TiB	da	Cunoscut pentru o viteză mai mare decât ext3-ul la lucrul cu multe fişiere de dimensiune mică
HFS+	Mac OS/ Linux	16 EiB	da*	* în Linux, HFS+ este suportat fără jurnalizare
ISO9660	Win/Linux/ Mac OS	În funcţie de implementar e	nu	Sistem de fişiere utilizat în principal pe CD-uri, organizare internă concepută pentru ca datele să fie citibile uşor
UDF	Win/Linux/ Mac OS	16 EiB	da	Sistem de fişiere utilizat în principal pe mediile optice, cu suport atât pentru scriere cât şi pentru citire

Figura 3-4 Caracteristici ale sistemelor de fişiere mai cunoscute

[8] *Ext2 este disponibil pe Windows folosind Ext2 Installable File System For Windows - http://www.fs-driver.org/*

3.1.3 Adresarea într-un sistem de fişiere

3.1.3.1 Numele fişierelor

Din punct de vedere fizic, fişierele ocupă pe dispozitivul de stocare un spaţiu bine definit. Utilizatorul nu lucrează direct cu dispozitivul de stocare şi nu poate adresa un fişier după poziţia fizică a acestuia pe dispozitivul de stocare. S-a introdus astfel adresarea fişierelor folosind nume. În funcţie de tipul sistemului de fişiere, există o serie de restricţii asupra acestor nume.

Majoritatea fişierelor au şi ceea ce se numeşte "extensie". Aceasta extensie se regăseşte după numele fişierelor şi este utilizată pentru identificarea tipului acestora – exemplu format: "nume.ext". În majoritatea cazurilor această extensie are 3 caractere. Sistemele de operare de tip Windows utilizează extensia pentru a asocia un tip de fişier cu o anumită aplicaţie care poate deschide acel tip de fişier. În mediile Unix, extensia de obicei lipseşte. Identificarea tipului fişierelor se realizează în general pe baza conţinutului fişierelor.

Sistem de fişiere	Case sensi-tive	Lungime nume	Lungime cale	Observaţii
FAT32	nu	"8.3" sau 255 caractere cu LFN	nedefinit	initial limitat la 8 caractere numele fişierului şi 3 caractere extensia, FAT32 a capatat suport pentru nume lungi Long Filename – LFN) – pentru un fişier, FAT32 reţine atât numele lung cât şi versiunea scurta în format 8.3
NTFS	da[9]	254 caractere + "."	32767 octeţi	Orice caracter Unicode poate fi folosit, mai puţin caracterele: NULL, · " / \ * ? < > | : (utilizate în general pentru adresarea fişierelor)
Ext3	da	255 octeţi	nedefinit	Orice byte mai puţin NULL poate fi folosit în numele fişierului
ReiserFS	da	4,032 octeţi/ 255 caractere	nedefinit	Orice octet mai puţin NULL poate fi folosit în numele fişierului
HFS+	nu[10]	255 caractere UTF-16	nedefinit	Orice caracter Unicode este permis.
ISO9660	nu	"8.3" sau Joliet: 128 octeţi	adâncime maximă a căii: 8 niveluri	Există mai multe niveluri pentru standard, fiecare cu restricţiile lui
UDF	da	255 octeţi	1023 octeţi	Orice octet mai putin NULL poate fi folosit în numele fişierului

Figura 3-5 Restricţii asupra numelor fişierelor

Comanda utilizată pentru a descoperi tipul fişierelor este:

```
file <nume_fisier>
```

[9] *Mediul Windows nu permite utilizarea de nume case-sensitive – alte medii precum Unix Services for Windows lucrează direct deasupra kernel-ului şi pot folosi nume case-sensitive.*

[10] *exista posibilitatea utilizării sistemului de fişiere cu nume case-sensitive (implicit: nu)*

Exemplu de utilizare:

```
  $ file /boot/kernel26.img
  /boot/kernel26.img: gzip compressed data, from Unix, last modified: Sat
Mar 3 20:28:37 2007, max compression
```

Numele fişierelor poate fi scris cu un anumit set de caractere. În prezent, majoritatea sistemelor de fişiere folosesc standardul Unicode (de obicei UTF-8 sau UTF-16) pentru codificarea caracterelor. Pentru mai multe informaţii despre Unicode. Mai jos sunt prezentate câteva sisteme de fişiere şi restricţiile de nume asociate lor.

Dupa cum s-a menţionat şi la începutul acestui capitol, directoarele se utilizează pentru o mai bună organizare a fişierelor. Într-un sistem de fişiere modern, fişierele şi directoarele se regăsesc într-o structură ierarhică - sub forma unui arbore. Acest arbore are un director părinte din care se poate ajunge în orice loc în sistemul de fişiere.

Calea către un fişier este formată dintr-o înşiruire de directoare, separate de caractere speciale. În mediile Unix separatorul este caracterul "/" (*slash*). În mediile Windows, separatorul este caracterul "\" (*backslash*).

3.1.3.2 Ierarhia sistemului de fişiere

Sistemele de fişiere permit utilizatorului să organizeze datele într-un mod accesibil. Structura cel mai des întâlnită pentru organizarea fişierelor este arborele. Tabelele de mai jos prezintă structura ierarhică din sistemele de operare cele mai cunoscute.

Director	Conţinut
/	directorul rădăcină
/bin	comenzi esenţiale necesare bootării, întreţinerii şi depanării sistemului
/boot	fişiere necesare bootării, precum imaginea kernel-ului
/dev	fişiere speciale utilizate pentru accesul direct la dispozitivele hardware sau logice ale sistemului
/etc	fişiere pentru configurarea sistemului, precum `inittab`, `fstab` şi `hosts`
/home	fişierele fiecărui utilizator din sistem – datele unui utilizator se găsesc în `/home/username`
/media	subdirectoare în care se montează unităţile optice, floppy etc.
/mnt	subdirectoare în care se montează alte sisteme de fişiere
/opt	pachete de aplicaţii de dimensiuni mari, accesibile tuturor utilizatorilor
/proc	sistem virtual de fişiere din care se obţin informaţii despre sistem şi aplicaţiile care rulează la un moment dat
/root	directorul home al utilizatorului root
/sbin	comenzi de bază accesibile numai utilizatorului root
/tmp	fişiere temporare
/usr	aplicaţii pentru uzul normal al sistemului de operare - `/usr/local` conţine aplicaţiile instalate/compilate de utilizator
/var	fişiere al căror conţinut se schimbă foarte des, precum log-uri, fişiere temporare, cache (date reutilizabile), spool (date neprocesate)

Figura 3-6 Ierarhia într-un sistem de fişiere din mediul Unix

Mai jos este se găseşte o reprezentare grafică parţială pentru o ierarhie dintr-un sistem de fişiere în Linux.

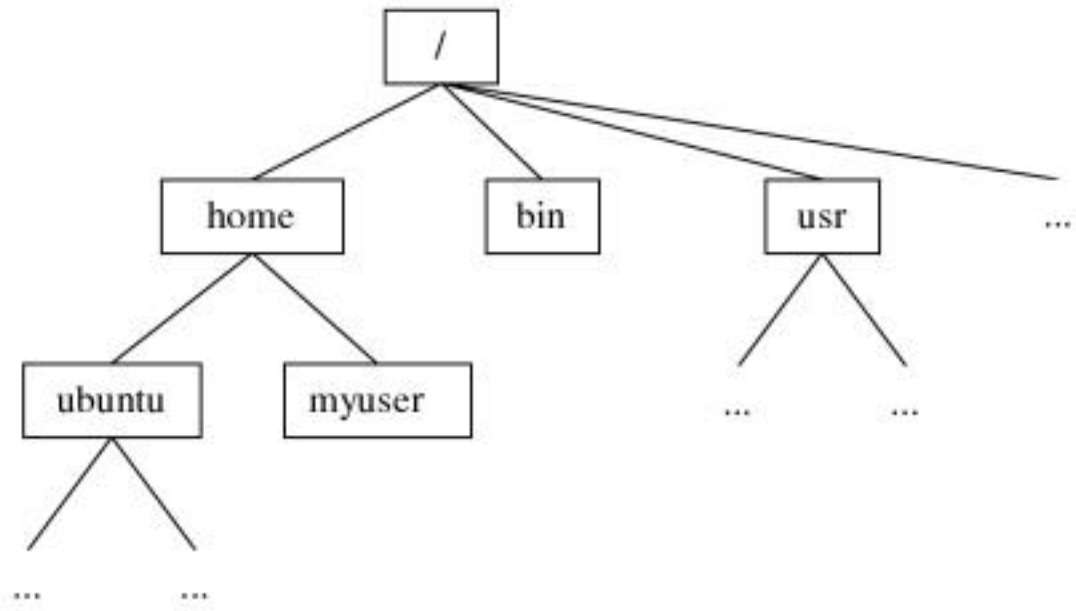


Figura 3-7 - Ierarhia sistemului de fişiere în Linux

În rădăcină se regăsesc directoarele `home`, `bin`, `usr` ..., în `home` se regăsesc directoarele `ubuntu` şi `myuser` ş.a.m.d.

Comparativ cu Linux-ul, structura în Windows este mult mai simplă pentru directoarele aflate imediat în rădăcină. În schimb, o mare parte din directoarele importante se ascund în directorul "Windows".

Director	Conţinut
`C:\`	directorul rădăcină
`C:\Windows`	Windows-ul şi fişierele aferente
`C:\Documents and Settings`	configurările utilizatorilor şi date specifice acestora
`C:\Program Files`	Aplicaţii
`C:\Windows\System32`	driver-e şi fişiere de configurare Windows
`C:\Documents and Settings\username\My Documents`	datele unui utilizator (aceasta este calea implicită, ea poate fi modificată)

Figura 3-8 Ierarhia într-un sistem de fişiere din Windows

Windows-ul are o particularitate: există mai multe directoare rădăcină, câte unul pentru fiecare sistem de fişiere – spre exemplu, pot exista simultan următoarele directoare rădăcină:

- `A:` - floppy disk (de obicei literele A: şi B: sunt rezervate de Windows pentru floppy disk-uri)
- `C:` - partiţie de pe hard disk; dacă există mai multe partiţii se asociază litere în ordine pentru fiecare dintre ele
- `D:` - `CD-ROM/DVD-ROM` (următoarea literă disponibilă după ce s-au asociat litere partiţiilor de pe hard disk(uri))

După cum s-a precizat anterior, pe o partiţie poate exista un singur sistem de fişiere la un moment dat. De obicei, sistemele de fişiere cu suport fizic se găsesc pe o singură partiţie (dintre exemplele menţionate anterior, doar UnionFS-ul nu se încadrează în aceasta categorie).

Trebuie menţionat faptul că Windows-ul alocă literele dupa partiţii, nu după sistemul de fişiere. Astfel, dacă sistemul de fişiere de pe o partiţie se schimbă, el va avea tot litera C:.

Tabelul următor prezintă o analogie între căile importante din sistemele de operare majore existente în prezent:

Descriere	Windows	Linux	Mac OS
radacină	`C:`	`/`	`/`
director home	`C:\Documents and Settings\ username`	`/home/username`	`/Users/username`
aplicații	`C:\Program Files`	`/bin; /sbin; /usr/bin; /usr/sbin; /usr/local/bin; /opt/*/bin`	`/Applications; /bin; /sbin`
configurări ale sistemului	Windows Registry	directoare specifice fiecărei aplicații, aflate în home-ul utilizatorului; `/etc`	`/Users/username/ Library; /etc`

Figura 3-9 Comparație între căile sistemelor de operare

3.1.3.3 Căi relative şi căi absolute

Există două moduri prin care se poate accesa un fişier sau un director: folosind o cale absolută sau o cale relativă.

Calea absolută este calea care începe cu directorul rădacină al sistemului de fişiere - în cazul Linux/Mac OS este "/"; în cazul Windows este "C:", "D:" etc.

Calea relativă este acea cale care indică spre un fişier pornind de la directorul curent.

> *Fiecare director conține două directoare speciale: „." (punct) şi „.." (două puncte).*
> *„." indică spre acelaşi director.*
> *„.." indică spre directorul părinte.*

Astfel, dacă se doreşte „urcarea" în ierarhia de fişiere, se foloseşte „.." care indică directorul părinte. Se poate urca mai mult în ierarhia de directoare înlănțuind câteva grupări de ".." separate prin separatorul de directoare.

Spre exemplu:

```
ubuntu@ubuntu:~/cs/uso$ pwd
/home/ubuntu/cs/uso
ubuntu@ubuntu:~/cs/uso$ cd ../../..
ubuntu@ubuntu:/home$ pwd
/home
```

Explicații:

- comanda `pwd` afişează directorul curent
- comanda `cd ../../..` schimbă directorul curent în parintele-parintelui-parintelui directorului curent (s-ar putea spune "străbunicul directorului curent")

„." se foloseşte pentru a indica în mod explicit o cale care are ca punct de pornire directorul curent. „." este utilizat frecvent pentru a scrie comenzi care execută script-uri/programe aflate în directorul curent.

Exemplu:

```
$ ./program_din_directorul_curent
```

execută fişierul "program_din_directorul_curent" - acest fişier trebuie să poată fi executat de utilizatorul curent.

Pentru mai multe exemple de utilizare a căilor relative şi absolute se poate consulta secțiunea 3.4.

3.1.3.4 Variabila de mediu PATH

În mod normal, pentru execuția programelor uzuale, nu este necesară specificarea căii complete sau schimbarea directorului. Sistemele de operare utilizează variabile de mediu (*environment variables*) pentru a stoca diferite date. PATH este o variabila de mediu care conține lista căilor uzuale unde se găsesc programe. Programele care se găsesc în oricare din căile din PATH pot fi executate fără a preciza calea către ele. Mai multe informații despre variabile de mediu şi modificarea acestora se regăsesc în secțiunea 11.4.1

Pentru a afişa variabila de mediu PATH într-un mediu Linux/Mac OS se utilizează comanda:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Căile din listă sunt despărțite prin separatorul „:".

Într-un mediu Windows, comanda este:

```
C:> echo %PATH%
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
```

3.2 Lucrul cu sistemele de fişiere

Acest subcapitol tratează acțiunile cele mai cunoscute care se fac asupra sistemelor de fişiere.

Acțiunile cele mai frecvente sunt:

- crearea unui sistem de fişiere
- montarea unui sistem de fişiere
- repararea unui sistem de fişiere

Printre acțiunile rare dar care pot reprezenta interes trebuie menționate:

- crearea unei imagini pentru un sistem de fişiere
- ştergerea unui sistem de fişiere (din diferite motive)

Pe lângă aceste acțiuni, folosind programe speciale, se pot efectua şi alte operații asupra unui sistem de fişiere, precum:

- redimensionarea unei partiții – duce implicit la o redimensionare a sistemului de fişiere (în cazul în care acest lucru nu se întâmpla, trebuie reformatată partiția)
- convertirea unui sistem de fişiere dintr-un tip în altul
- mutarea unei partiții - de obicei nu are efecte asupra sistemului de fişiere aflat pe aceasta

3.2.1 Crearea unui sistem de fişiere

După cum s-a menționat anterior, majoritatea sistemelor de fişiere au un suport fizic unde se regăsesc efectiv datele. Suportul unui sistem de fişiere se numeşte partiție. O partiție poate avea un singur sistem de fişiere. Partiționarea a fost descrisă în subcapitolul 2.2.

Sistemul de fişiere este plasat pe o partiție prin formatare. Formatarea este procesul prin care pe partiție sunt scrise structuri de date specifice sistemului de fişiere. Aceste structuri sunt utilizate de driver-ul asociat sistemului de fişiere pentru a identifica şi utiliza datelor scrise pe partiție.

Într-un sistem Linux, formatarea se realizează prin intermediul utilitarului mkfs. Acesta se apelează în felul următor:

```
# mkfs -t <tip_sistem_de_fişiere> <dispozitiv>
```

Exemplu de utilizare:

```
# mkfs -t ext3 /dev/sda2
```

Comanda prezentata anterior formatează partiţia a 2-a de pe primul disc utilizând sistemul de fişiere ext3. Pentru fiecare tip de sistem de fişiere exista un program pentru formatare. Aceste programe au numele de forma `mkfs.<fstype>` şi pot fi apelate direct pentru a formata o partiţie. Comanda echivalentă exemplului anterior este următoarea:

```
# mkfs.ext3 /dev/sda2
```

3.2.2 Montarea unui sistem de fişiere

Sistemele de operare oferă acces uniform la datele din toate tipurile de sisteme de fişiere prin intermediul ierarhiei de fişiere. Pentru ca un sistem de fişiere să fie vizibil sistemului de operare el trebuie întâi să fie „montat" (*mount*) într-un director, numit director de montare (*mount point*). Aceasta operaţie este executată de cele mai multe ori în mod automat de sistemul de operare:

În Windows, sistemele de fişiere primesc automat litere diferite (pot fi ulterior modificate, cu anumite restricţii, folosind `Control Panel > Administrative Tools > Computer Management > Storage > Disk Management`)

În Linux, sistemele de fişiere sunt montate atât în locuri bine definite (definite de utilizator la instalare) cât şi în locuri special create pentru a găzdui sistemele de fişiere de pe medii externe, precum USB stick-uri, CD/DVD-ROM-uri etc., în directoare precum `/media/<nume>`.

În Mac OS, sistemele de fişiere sunt montate automat în `/Volumes/<nume>`.

3.2.2.1 Montarea manuală

Pentru a monta un sistem de fişiere, într-un mediu Linux se utilizează comanda `mount` , cu următorii parametri:

```
# mount -t <tip_sistem_fişiere> [-o <opţiuni>] <dispozitiv>
<director_montare>
```

Exemplu de utilizare a comenzii `mount`:

```
# mount -t ext3 /dev/sda2 /mnt/data
```

montează partiţia a 2-a de pe primul disc, cu sistemul de fişiere ext3, în directorul /mnt/data (directorul /mnt/data trebuie să existe anterior execuţiei comenzii).

Exemplu de utilizare a opţiunilor de montare se găseşte în continuare, în subcapitolul de studii de caz.

3.2.2.2 Montarea automată

Într-un sistem Linux, fişierul `/etc/fstab` descrie sistemele de fişiere care vor fi montate automat la pornirea sistemului de operare. Fisierul conţine câte o linie pentru fiecare sistem de fişiere. Comentariile încep cu #.

Sintaxa unei linii este următoarea:

```
<dispozitiv> <director montare> <tip sistem fişiere> <opţiuni>
<dump>  <pas>
```

Câmpurile au următoarele semnificaţii:

- <dispozitiv> - fişier care indica locul (de obicei un dispozitiv) unde se gaseste sistemul de fişiere
- <director montare> - director existent în structura de directoare unde se va putea accesa conţinutul sistemului de fişiere montat
- <tip sistem fişiere> - tipul sistemului de fişiere

- <opțiuni> - opțiuni pentru montarea sistemului de fișiere – diferă de la un sistem de fișiere la altul
- <dump> - indică dacă să se realizeze automat backup al partiției utilizând programul dump - în prezent nu prea este folosită opțiunea, preferându-se alte metode de backup; acest parametru are în mod normal are valoarea 0
- <pas> - indică ordinea în care fsck va verifica sistemele de fișiere (0 = fsck nu va verifica sistemul de fișiere)

Un exemplu de linie tipică din `/etc/fstab` este următoarea:

```
/dev/sda1                /               reiserfs defaults 0 0
```

Linia descrie montarea primei partiții de pe primul disc în /, utilizând sistemul de fișiere *reiserfs*, opțiuni implicite, fără backup și fără verificări.

După efectuarea de modificări în `/etc/fstab`, comanda

```
# mount -a
```

va monta toate partițiile menționate în acest fișier.

Comanda `mount` fără nici un parametru afișează lista sistemelor de fișiere montate la un moment dat.

```
$ mount
/dev/sda1 on / type reiserfs (rw)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
none on /dev/pts type devpts (rw)
none on /dev/shm type tmpfs (rw)
tmpfs on /tmp type tmpfs (rw)
```

Se observă că în listă nu apar doar sistemele de fișiere montate manual sau cele montate automat (descrise în `/etc/fstab`). Există și sisteme de fișiere care sunt create și montate automat de sistem în ierarhia de fișiere pentru a oferi informații despre starea sistemului (precum `/proc` [vezi 6.7]).

Lista sistemelor de fișiere montate la un moment dat se regăsește și în fișierul `/etc/mtab`.

Operația inversă montării unui sistem de fișiere se numește demontare. Pentru a realiza demontarea se utilizează comanda `umount`, cu unul din următoarele formate:

```
# umount <dispozitiv>
# umount <director_de_montare>
```

Exemplu de utilizare. Cunoscând că:

```
# mount | grep /dev/sda2
/dev/sda2 on /home type reiserfs (rw)
```

Pentru a demonta acest sistem de fișiere se poate utiliza oricare din următoarele două comenzi:

```
# umount /dev/sda2
# umount /home
```

3.2.3 Repararea unui sistem de fișiere

După cum s-a menționat și la începutul acestui capitol, sistemele de fișiere moderne au suport pentru jurnalizare. Acest suport este util în mod deosebit atunci când se produce o cădere a sistemului, precum atunci când se întrerupe alimentarea cu energie electrică.

La repornire, sistemul de operare efectuează o verificare a integrității sistemelor de fișiere. La această verificare, se execută comanda `fsck` pentru fiecare sistem de fișiere. Aceasta

comanda este particularizata pentru fiecare sistem de fişiere cu numele `fsck.<tip_sistem_de_fisiere>`, precum comanda `mkfs`.

Pentru a verifica un sistem de fişiere comanda `fsck` se apeleaza în felul următor:

```
# fsck -t <tip_sistem_fişiere> <dispozitiv>
```

Dacă se doreşte verificarea celei de-a doua partiţii de pe primul disc, cunoscându-se că ar trebui să se găsească un sistem de fişiere ext3 pe aceasta, se utilizează comanda:

```
# fsck -t ext3 /dev/sda2
```

În funcţie de opţiunile care se folosesc la execuţia comenzii, `fsck` poate să şi repare erorile pe care le întâlneşte. Multe detalii despre utilizarea lui `fsck` se regăsesc în `man fsck`.

3.2.4 Crearea unei imagini pentru un sistem de fişiere

Imaginea unui sistem de fişiere reprezintă copia fidelă într-un fişier a fiecărui octet care descrie sistemul de fişiere (inclusiv datele din acesta). Această imagine poate fi ulterior copiată pe o altă partiţie sau arhivată pentru o eventuală recuperare a datelor dacă partiţia sursă este distrusă accidental. O utilizare foarte frecventă a imaginilor se întâlneşte pentru a copia conţinutul CD/DVD-urilor pe alte medii de stocare.

Pentru a realiza o imagine a unui sistem de fişiere se utilizează comanda `dd` [vezi 6.6.2]

Formatul comenzii utilizate este următorul:

```
# dd if=<dispozitiv> of=<fisier>
```

Exemplu de utilizare:

```
# dd if=/dev/sdb1 of=/mnt/data/partitie1.img
```

copiază conţinutul primei partiţii (sistemul de fişiere) de pe discul al 2-lea în fişierul `/mnt/data/partitie1.img`.

Această comandă este independentă de sistemul de fişiere căruia i se face imaginea. Ea lucrează direct la nivelul octeţilor care alcătuiesc sistemul de fişiere şi păstrează întreaga structură internă a acestuia.

În cazul utilizării comenzii pe sisteme de fişiere foarte mari, se va observa că timpul de execuţie este destul de mare. Motivul pentru care se întâmplă acest lucru este faptul că, în mod implicit, comanda `dd` lucrează cu blocuri de date de 512 octeţi, ceea ce înseamnă ca pentru fiecare 512 octeţi comanda execută o citire şi o scriere. În prezent, buffer-ele discurilor au valori dimensiuni mult mai mari şi pot lucra cu blocuri mai mari de 512 octeţi.

Pentru a modifica dimensiunea blocului citit/scris de `dd` se utilizează parametrul `bs=<dimensiune>`, unde `<dimensiune>` poate sa fie „xKB", „xMB" etc.

Spre exemplu:

```
# dd bs=2MB if=/dev/sdb1 of=/mnt/data/partitie1.img
```

va executa aceeaşi operaţie dar citind blocuri de 2MB.

Există şi un dezavantaj al utilizării unor blocuri de date mai mari. În anumite condiţii, nu se va putea face copierea unui bloc întreg, din diferite motive precum un defect hardware al suprafeţei magnetice, sau ultimul bloc nu are dimensiunea de 2MB (deoarece dimensiunea totală nu este multiplu de 2MB). De cele mai multe ori se doreşte recuperarea cât mai multor date de pe disc, dacă nu chiar toate. În aceste condiţii trebuie determinat daca o viteza mai mare (dimensiune mai mare a blocului) este o decizie acceptabila ştiind ca se pot pierde date mai multe la citire.

Pentru recuperarea datelor de pe discuri cu defecte hardware, există programul `ddrescue`[1] - `ddrescue` este o aplicație asemănătoare `dd`-ului dar care este specializată în recuperarea cât mai multor date de pe discuri. Mai multe informații se găsesc pe pagina proiectului `ddrescue`.

3.2.5 "Ştergerea" unui sistem de fişiere

Exista două tipuri de ştergere a unui sistem de fişiere:

- ştergerea logică
- ştergerea fizică

Ştergerea logică se obține prin ştergerea partiției corespunzătoare sistemului de fişiere din tabela de partiții (precum a fost descris în secțiunea 2.2. Din moment ce nu mai există partiție care să încapsuleze sistemul de fişiere, acesta se poate considera şters. Practic, datele lui încă mai rezidă pe disc, până la o suprascriere cu alte date.

Faptul ca datele nu sunt şterse fizic de pe disc a pus şi încă pune probleme oamenilor care doresc ca datele o dată ce sunt şterse să rămână şterse pentru eternitate. Din acest motiv, pentru a elimina orice posibilitate de a recupera un sistem de fişiere sau datele dintr-un sistem de fişiere, se utilizează ştergerea fizică. Ştergerea fizică este realizată de fapt prin suprascrierea blocurilor de pe suportul fizic (unde era înainte sistemul de fişiere) cu date aleatorii sau cu zero-uri.

Pentru suprascriere se utilizează cel mai rapid tot programul `dd` prezentat anterior dar, în loc ca partiția să fie folosită ca sursă, ea este folosită ca destinație pentru un set de date aleatoare (sau zero-uri).

Pentru a suprascrie cu zero-uri o partiție, se poate folosi comanda:

```
dd if=/dev/zero of=/dev/sda1
```

Pentru a suprascrie o partiție cu date aleatoare, se pot scrie blocuri aleatoare pe partiție, folosind comanda:

```
dd if=/dev/urandom of=/dev/sda1
```

Mai multe informații despre comanda `dd` se pot afla în secțiunea 6.6.2.

3.3 Tipuri de fişiere

3.3.1 Terminologie

Sistemele de fişiere pun la dispoziție utilizatorilor diferite forme logice de organizare şi acces la date, bineînțeles sub formă de fişiere. Intern, pentru a prezenta datele utilizatorului, sistemele de fişiere lucrează cu entități de două feluri:

- dentry – reține numele asociat unui inode (un fel de "pointer" către inode)
- inode – reține unde se găsesc datele dintr-un fişier (practic, inode-ul este structura asociată unui fişier); inode-ul reține câte dentry-ui indică la el

În afară de directoare şi fişiere sistemul de fişiere pune la dispoziția utilizatorului şi legături (*link*-uri). Acestea pot fi de două tipuri: legături simbolice şi legături hardware**. Legăturile simbolice** au avantajul că pot fi făcute între sisteme de fişiere diferite. Ele pot reține calea relativă între nume şi fişierul/directorul destinație. Sunt uşor identificabile, deoarece au un tip special. Defectul major e faptul că în momentul în care destinația este mutată, legătură trebuie actualizată. În cazul în care destinația este reprezentată ca o cale relativă, mutarea legăturii duce automat la invalidarea ei.

---

[1] *http://www.gnu.org/software/ddrescue/ddrescue.html*

Legăturile hard au avantajul că sunt complet transparente utilizatorului. Sistemul de fişiere numără câte nume indică un anume conţinut şi în momentul în care nu mai există nume se consideră că fişierul a fost şters. Dezavantajul legăturilor hard este ca pot fi utilizate numai în acelaşi sistem de fişiere (un nume nu poate indica un conţinut dintr-un alt sistem de fişiere). Din moment ce legăturile sunt transparente utilizatorului, există posibilitatea ca la copierea fişierelor de pe un sistem de fişiere cu multe legături hard să se obţină multe duplicate pentru aceleaşi date.

Astfel, sistemele de fişiere moderne oferă posibilitatea utilizării mai multor tipuri de fişiere:

Tipuri fişier	**Denumire tip în engleză**	**Descriere**
fişiere normale	files	unitate logică de acces la date
directoare	directories/folders	organizează fişiere şi alte directoare
legaturi simbolice	symbolic links	legatura către un alt fişier – legatura este retinuta într-un nou inode; numele legaturii este pastrat într-un nou dentry (care indica la noul inode)
legături hard	hard links	legatura către un alt fişier – un nou dentry este creat; inca la inode-ul initial; inode-ului i se incrementeaza numarul de referinte către sine
pipe-uri cu nume	named pipes	fişiere folosite pentru transferuri FIFO (First în-First Out) între două procese: un proces deschide fişierul pentru scriere, alt proces deschide fişierul pentru citire; ceea ce este scris poate fi citit simultan

Există posibilitatea de a crea legături atât pentru fişiere cât şi pentru directoare – în fond, ambele entităţi sunt fişiere (dar cu atribute diferite). Legăturile se utilizează la scriere/citire precum fişierele normale. Sistemul de operare împreună cu sistemul de fişiere se ocupă transparent de „traducerea" acestora în fişiere normale pentru aplicaţii.

Exemplul următor prezintă utilizarea legăturilor simbolice şi a celor hard. Comanda `ls -l` este utilizată pentru a lista fişierele detaliat.

```
[35]ubuntu@ubuntu:~$ ls -l
total 4
drwxr-xr-x 3 ubuntu ubuntu 100 2007-09-04 10:56 cs
drwxr-xr-x 2 ubuntu ubuntu 100 2007-09-03 13:15 Desktop
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
[36]ubuntu@ubuntu:~$ ln -s /var/log/kern.log
[37]ubuntu@ubuntu:~$ ls -l
total 4
drwxr-xr-x 3 ubuntu ubuntu 100 2007-09-04 10:56 cs
drwxr-xr-x 2 ubuntu ubuntu 100 2007-09-03 13:15 Desktop
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
lrwxrwxrwx 1 ubuntu ubuntu  17 2007-09-04 10:59 kern.log -> /var/log/kern.log
```

Comanda `ln -s` de mai sus a detectat automat ce nume să folosească pentru legătura simbolica către `/var/log/kern.log`.

Se poate utiliza un nume diferit pentru legătura simbolică, precum mai jos:

```
[38]ubuntu@ubuntu:~$ ln -s hello.txt hello_link_symbolic.txt
[39]ubuntu@ubuntu:~$ ls -l
total 4
drwxr-xr-x 3 ubuntu ubuntu 100 2007-09-04 10:56 cs
drwxr-xr-x 2 ubuntu ubuntu 100 2007-09-03 13:15 Desktop
lrwxrwxrwx 1 ubuntu ubuntu   9 2007-09-04 11:00 hello_link_symbolic.txt -> hello.txt
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
lrwxrwxrwx 1 ubuntu ubuntu  17 2007-09-04 10:59 kern.log -> /var/log/kern.log
[39]ubuntu@ubuntu:~$ rm kern.log
```

Legăturile hard se realizează cu aceeaşi comanda, dar fără parametrul `-s`, precum în exemplul următor:

```
[40]ubuntu@ubuntu:~$ ln hello.txt hello_link_hard.txt
[41]ubuntu@ubuntu:~$ ls -l
total 8
drwxr-xr-x 3 ubuntu ubuntu 100 2007-09-04 10:56 cs
-rw-r--r-- 2 ubuntu ubuntu  37 2007-09-04 10:59 hello_link_hard.txt
lrwxrwxrwx 1 ubuntu ubuntu   9 2007-09-04 11:00 hello_link_symbolic.txt -> hello.txt
-rw-r--r-- 2 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
```

Se observă ca numărul de legături pentru fişierele hello.txt şi hello_link_hard.txt este 2, căci ele sunt fişierele care indică acelaşi conţinut. Pentru demonstraţie, se înlocuieşte conţinutul unui fişier.

```
[42]ubuntu@ubuntu:~$ echo "Hello" > hello_link_hard.txt
[43]ubuntu@ubuntu:~$ ls -l
total 8
drwxr-xr-x 3 ubuntu ubuntu 100 2007-09-04 10:56 cs
-rw-r--r-- 2 ubuntu ubuntu   6 2007-09-04 11:01 hello_link_hard.txt
lrwxrwxrwx 1 ubuntu ubuntu   9 2007-09-04 11:00 hello_link_symbolic.txt -> hello.txt
-rw-r--r-- 2 ubuntu ubuntu   6 2007-09-04 11:01 hello.txt
```

Se observă mai sus că după modificare atât dimensiunea cât şi data ambelor fişiere ce referă acelaşi conţinut s-a modificat. Dimensiunea fişierului cu legătura simbolică nu s-a modificat.

Ştergerea fişier care indică aceleaşi conţinut cu altul va face să scadă numărul de legături hard pentru fişierul rămas.

```
[44]ubuntu@ubuntu:~$ rm hello.txt
[45]ubuntu@ubuntu:~$ ls -l
total 4
drwxr-xr-x 3 ubuntu ubuntu 100 2007-09-04 10:56 cs
-rw-r--r-- 1 ubuntu ubuntu   6 2007-09-04 11:01 hello_link_hard.txt
lrwxrwxrwx 1 ubuntu ubuntu   9 2007-09-04 11:00 hello_link_symbolic.txt -> hello.txt
```

Pipe-urile cu nume sunt utilizate în principal pentru comunicaţia inter-procese [vezi 4.5]. Se regăsesc sub formă de fişiere goale, cu un atribut special - „p", vizibil pe primă coloană în ieşirea comenzii `ls -l`. Un proces scrie în pipe în timp ce alt proces citeşte din pipe.

```
$ mkfifo fifo1
$ ls -l
total 0
prw-r--r-- 1 mircea users  0 2007-09-04 11:10 fifo1
```

3.3.2 Detecţia tipului fişierelor

Pe sistemele Windows, detecţia tipului fişierelor se bazează pe verificarea extensiei. În funcţie de aceasta, sistemul de operare determină care aplicaţie este potrivită pentru a deschide un fişier.

Pe sistemele Linux, detecţia tipului fişierelor este realizată prin intermediul comenzii `file`. Aceasta utilizează un fişier special de configurare în care sunt înscrise secvenţe de octeţi care se regăsesc în diferite tipuri de fişiere. `file` lucrează independent de extensia fişierului, precum este dovedit în exemplele următoare:

```
$ file photo.jpg
photo.jpg: JPEG image data, JFIF standard 1.01
$ mv photo.jpg fisier.txt
$ file fisier.txt
photo.txt: JPEG image data, JFIF standard 1.01
```

3.4 Operaţii uzuale asupra fişierelor şi directoarelor

3.4.1 Afişarea şi schimbarea directorului curent

Comanda `pwd` afişează calea absolută către directorul curent. De asemenea, prompt-ul bash conţine implicit directorul curent. Pentru a schimba directorul curent se foloseşte comanda: `cd <cale>`. Calea poate să fie o cale absolută sau relativă.

Mai jos se găseşte un exemplu de folosire a comenzilor `pwd` şi `cd` - a se observa că la fiecare schimbare de director se modifică şi prompt-ul bash. Ierarhia de directoare prin care se navighează este cea prezentată anterior.

```
[5]ubuntu@ubuntu:~$ pwd
/home/ubuntu
[6]ubuntu@ubuntu:~$ cd ..
[7]ubuntu@ubuntu:/home$ pwd
/home
[8]ubuntu@ubuntu:/home$ cd ../usr/bin
[9]ubuntu@ubuntu:/usr/bin$ pwd
/usr/bin
[10]ubuntu@ubuntu:/usr/bin$ cd .
[11]ubuntu@ubuntu:/usr/bin$ cd /
[12]ubuntu@ubuntu:/$ cd
[13]ubuntu@ubuntu:~$ pwd
/home/ubuntu
[14]ubuntu@ubuntu:~$ cd /usr/bin
[15]ubuntu@ubuntu:/usr/bin$ cd /home
[16]ubuntu@ubuntu:/home$ cd -
/usr/bin
[17]ubuntu@ubuntu:/usr/bin$ cd ~
[18]ubuntu@ubuntu:~$ pwd
/home/ubuntu
[19]ubuntu@ubuntu:~$ cd ./././.
[20]ubuntu@ubuntu:~$ cd ../..
[21]ubuntu@ubuntu:/$ pwd
/
[22]ubuntu@ubuntu:/$
```

Observaţii:

- `cd ..` (comanda 8) schimbă directorul în directorul părinte
- „`.`" este o referinţă la directorul curent, deci `cd .` nu modifică directorul
- `~` este echivalentul directorului home pentru utilizatorul curent (comanda 11)
- `cd` apelat fără parametri (comanda 12) schimbă directorul în directorul home
- `cd -` - schimbă directorul în directorul anterior

3.4.2 Afişarea conţinutului fişierelor

Conţinutul unui fişier poate fi afişat pe ecran prin intermediul comenzii `cat`. Formatul acestei comenzi este: `cat <nume fişier>`. Un exemplu de utilizare este dat în continuare.

```
$ cat /etc/resolv.conf
# Generated by dhcpcd for interface eth0
search localdomain
nameserver 192.168.0.1
```

Această comandă afişează tot fişierul, chiar dacă acesta este mai mare de un ecran. Pentru a putea naviga prin output-ul unei comenzi (în cazul acesta particular – comanda cat), se pot folosi comenzile more sau less.

`more` permite navigarea doar intr-o singură direcție (de la început către sfârșitul fișierului) și câte un ecran o dată, pe când `less` permite navigarea în ambele direcții, câte o linie, precum într-un fișier. Interfața lui `less` este foarte asemănătoare cu cea a editorului `Vim`. În continuare sunt prezentate exemple de utilizare pentru aceste comenzi:

```
$ cat /etc/X11/xorg.conf | more
$ cat /etc/X11/xorg.conf | less
```

3.4.3 Listarea fișierelor

Cea mai frecventă operație care se execută asupra fișierelor este listarea. Comanda utilizată este `ls`. În această secțiune, prin fișier înțelegem orice tip de fișier (inclusiv tipul „director"), mai puțin în cazurile când este menționat explicit.

Formatul comenzii pentru listare este: `ls [opțiuni] [<cale>]`. În momentul în care lipsește calea, se realizează listarea conținutului directorului curent.

În continuare sunt prezentate opțiunile cele mai des folosite cu aceasta comandă:

`-l` = afișează informații detaliate despre fiecare fișier/director (data modificare, dimensiune, utilizator, grup, drepturi de acces)

`-a` = afișează și fișierele care încep cu "." (caracterul punct) – în Linux, aceste fișiere sunt considerate de majoritatea interfețelor cu utilizatorul ca fiind fișiere ascunse

`-h` = afișează dimensiunea fișierelor în format *human-readable*, respectiv dimensiunea în octeți este înlocuită cu dimensiunea în Kocteți/Mocteți/Gocteți dacă depășește un anumit ordin de mărime.

Formatul utilizat la afișarea detaliată este următorul:

```
$ ls -l /home/myuser/myapp
-rwxr----- 1 myser users 1176348 2007-03-03 20:28 /home/myuser/myapp
```

cu următoarele coloane:

- primul caracter reprezintă tipul fișierului - = fișier normal, d = director, p = pipe, b = dispozitiv bloc, c = dispozitiv caracter, l = legătură simbolică
- urmează 3 grupuri de 3 caractere (rwx) care reprezintă drepturile de acces pentru utilizatorul myuser (care este owner-ul fisierului), drepturile de acces pentru utilizatorii care fac parte din grupul „users" (din care face parte fișierul), drepturile de acces pentru utilizatorii care nu sunt „myuser" și nici nu fac parte din grupul „users" – mai multe detalii despre drepturile de acces în subcapitolul [vezi 3.6]
- numărul de link-uri hard către fișier - 1
- posesorul fișierului (en: *owner*) – *myuser*
- grupul de care aparține fișierul – *users*
- dimensiunea fișierului – 1176348
- data fișierului - 2007-03-03
- ora fișierului – 20:28
- numele fișierului - /home/myuser/myapp

Exemplele următoare prezintă câteva utilizări mai întâlnite pentru comanda `ls` care combină parametri prezentați anterior:

```
[34]ubuntu@ubuntu:~$ ls
Desktop
[35]ubuntu@ubuntu:~$ ls ~
Desktop
[36]ubuntu@ubuntu:~$ ls -l ~
total 0
drwxr-xr-x 2 ubuntu ubuntu 100 2007-09-03 13:15 Desktop
[38]ubuntu@ubuntu:~$ ls -alh
total 56K
drwxr-xr-x 7 ubuntu ubuntu  580 2007-09-03 19:58 .
drwxr-xr-x 3 root   root     60 2007-09-03 06:14 ..
-rw------- 1 ubuntu ubuntu   34 2007-09-03 19:58 .bash_history
-rw-r--r-- 1 ubuntu ubuntu 2.3K 2007-09-03 06:14 .bashrc
drwxr-xr-x 2 ubuntu ubuntu  100 2007-09-03 13:15 Desktop
-rw-r--r-- 1 ubuntu ubuntu  566 2007-09-03 06:14 .profile
-rw------- 1 ubuntu ubuntu 9.5K 2007-09-03 17:40 .xsession-errors
[39]ubuntu@ubuntu:~$ ls ../../
bin    dev   initrd      media  proc  sbin  tmp  vmlinuz
boot   etc   initrd.img  mnt    rofs  srv   usr
cdrom  home  lib         opt    root  sys   var
[40]ubuntu@ubuntu:~$ ls -a /
.    boot   etc     initrd.img  mnt   rofs  srv  usr
..   cdrom  home    lib         opt   root  sys  var
bin  dev    initrd  media       proc  sbin  tmp  vmlinuz
```

Observaţii:

- dacă se doreşte afişarea conţinutului directorului curent, nu este necesară scrierea căii ca parametru pentru comanda `ls`
- comanda 37 afişează lista detaliată a fişierelor din directorul home al utilizatorului curent, inclusiv fişierele ascunse (a se observă că şi „.” şi „..” sunt afişate
- comanda 38 afişează lista detaliată a fişierelor din directorul home al utilizatorului curent, utilizând formatul "*human-readable*" pentru afişarea dimensiunii fişierelor (a se observa dimensiunea fişierului .xsession-errors)
- comanda 39 prezintă utilizarea lui `ls` cu parametru dat sub formă de cale relativă (directorul afişat este /)
- comanda 40 prezintă fişierele ascunse ale directorului / - se observă că şi acest director are directoarele standard „.” şi „..” în el.

Pentru mai multe opţiuni se poate consulta `ls --help` sau `man ls`.

3.4.4 Crearea fişierelor/directoarelor

Pentru a crea orice entitate pe un sistem de fişiere, se utilizează o serie de comenzi.

Entitate	Comandă
Fişier normal	`touch <nume_fişier>` `> <nume_fişier>`
Director	`mkdir <nume_director>`
Legaturi simbolice	`Ln -s <destinatie>` `[<nume_legătură>]`
Legaturi hard	`ln <destinatie> [<nume_legătură>]`
Pipe-uri cu nume	`mkfifo <nume_pipe>`

Figura 3-10 Comenzi pentru crearea fişierelor

Primul mod de creare a unui fişier este utilizând comand comanda `touch`. A doua metodă se bazează pe o funcţionalitate `bash` numită redirectare în fişiere (va fi discutată în subcapitolul

următor). Pe scurt, ceea ce realizează comanda a 2-a este redirectarea ieşirii unei comenzi (nule) către un fişier, creându-se astfel un fişier gol.

```
$ touch fisier1
$ > fisier2
$ mkdir dir1
$ mkfifo fifo1
$ ls -l
total 0
drwxr-xr-x 2 mircea users 40 2007-09-20 10:26 dir1
prw-r--r-- 1 mircea users  0 2007-09-20 10:30 fifo1
-rw-r--r-- 1 mircea users  0 2007-09-20 10:26 fisier1
-rw-r--r-- 1 mircea users  0 2007-09-20 10:26 fisier2
```

Exemplele următoare prezintă crearea legăturilor pe scurt:

```
$ ln fisier1 hardlink
$ ln -s fisier2 softlink
$ ls -l
total 0
drwxr-xr-x 2 mircea users 40 2007-09-20 10:26 dir1
prw-r--r-- 1 mircea users  0 2007-09-20 10:30 fifo1
-rw-r--r-- 2 mircea users  0 2007-09-20 10:26 fisier1
-rw-r--r-- 1 mircea users  0 2007-09-20 10:26 fisier2
-rw-r--r-- 2 mircea users  0 2007-09-20 10:26 hardlink
lrwxrwxrwx 1 mircea users  7 2007-09-20 10:32 softlink -> fisier2
```

Se observă faptul ca atât legăturile simbolice cât şi legăturile hard se realizează cu aceeaşi comandă, respectiv comanda `ln`. Singura diferenţă între cele două situaţii este prezenţa parametrului `-s` în cazul creării legăturilor hard.

3.4.5 Copiere/mutare/redenumire/stergere

Observaţie: în cadrul comenzilor prezentate în continuare `<sursa>` poate fi reprezentată dată printr-o expresie regulată.

Copierea şi mutarea sunt operaţii care necesită 2 parametri:

- sursă de unde se copiază/mută
- destinaţia unde se copiază/mută

3.4.5.1 Copierea

Printre opţiunile mai folosite:

`-R` - copiere recursivă (copiază şi copii directoarelor, presupunând că există aşa ceva în sursă)

`-a` - copiere cu păstrare a tuturor atributelor (permisiuni, dată)

Format comanda:

```
cp [opţiuni] <sursa> <destinatie>
```

```
$ touch f1.txt
$ ls
f1.txt
$ cp f1.txt f2.txt
$ ls
f1.txt  f2.txt
```

3.4.5.2 Mutarea

Format comanda:

```
mv [opţiuni] <sursa> <destinatie>
```

Implicit, mutarea este recursivă şi păstrează atributele fişierelor – efectiv, la o mutare, se schimbă doar părintele fişierului sau directorului care se mută.

```
$ pwd
/tmp/q
$ ls
f1.txt  f2.txt
$ mv f1.txt /tmp/r
$ ls
f2.txt
$ ls /tmp/r/
f1.txt
```

3.4.5.3 Redenumirea

Redenumirea se poate realiza atât cu `cp` cât şi cu `mv`, în următoarele conditii

- în cazul în care sursa este un `fişier` şi destinaţia este un `fişier`, se realizează copierea/mutarea fişierului cu schimbarea numelui sursei
- în cazul în care sursa este *un director* şi destinaţia nu *exista ca director*, se realizeaza copierea/mutarea directorului cu schimbarea numelui

```
$ ls
f2.txt
$ mv f2.txt f3.txt
$ ls
f3.txt
```

3.4.6 Stergerea fişierelor/directoarelor

În Linux, comanda cea mai utilizata pentru ştergerea fişierelor şi directoarelor este `rm` (poate primi ca parametru şi o expresie regulată – [vezi 11.7.2]).

Formatul comenzii este:

```
rm [opţiuni] <cale>
```

Una dintre cele mai folosite opţiuni este -r/-R, utilizata pentru a şterge recursiv un director. Aceasta opţiune trebuie folosita cu atenţie pentru ca poate avea, spre exemplu, rezultate negative dacă directorul care este şters conţine multe informaţii utile. O eroare destul de întâlnită este utilizarea comenzii `rm -rf /`, care şterge recursiv totul începând cu rădăcina, forţând ştergerile (datorită lui `-f`).

Pentru ştergerea directoarelor goale se mai poate folosi şi `rmdir`, cu formatul:

```
rmdir <nume_director_gol>.
```

3.4.7 Arhivarea fişierelor şi dezarhivarea

Arhivarea este procedeul prin care mai multe fişiere şi directoare sunt adunate la un loc (într-un singur fişier), realizând eventual şi o reducere a dimensiunii prin eliminarea datelor care se repetă şi înlocuirea lor cu o serie de codificări.

În principiu, se creează un dicţionar cu secvenţe de octeţi frecvente şi o codificare a lor mai scurtă (pe mai puţini octeţi). Arhiva conţine atât dicţionarul cât şi conţinutul fişierelor dar, în loc de secvenţele frecvente lungi, se regăsesc referinţe către dicţionar. Dacă fişierele conţin multe date care se repetă, dimensiunea unei arhive scade considerabil.

După cum se poate identifica şi din descrierea anterioară, există două etape mai importante în ceea ce priveşte manipularea fişierelor:

- concatenarea (lipirea) fişierelor într-un fişier mai mare, din care să se poată extrage toate fişierele şi informaţiile despre ele
- compresia fişierului mare, astfel încât să se reducă dimensiunea lui dar fără a se pierde din informaţii

În Linux există utilitare de arhivare care se ocupă ori de una dintre etape ori de ambele etape.

Comanda `tar` se ocupă de prima etapă. Iniţial, rezultatul comenzii `tar` era transferat pe benzi magnetice, din acest motiv numele utilitarului vine de la *tape archiver*.

Un fişier *.tar* conţine necompresat fişiere împreună cu informaţii despre modul de extragere al acestora (spre exemplu: de unde până unde se găseşte un fişier în cadrul arhivei). Din acest motiv, un fişier *.tar* este de obicei un pic mai mare decât suma tuturor dimensiunilor fişierelor ce sunt incluse în el.

Pentru a crea o arhiva tar, se utilizează comanda:

```
$ tar cvf nume_arhiva.tar <cale>
```

cu parametri:

- `c`: crează arhiva
- `v`: afişează ce anume se arhivează
- `f nume_arhiva.tar`: specifică numele arhivei
- `<cale>`: ce anume se arhivează

Pentru a dezarhiva o arhiva .tar, se utilizeaza:

```
$ tar xvf nume_arhiva.tar [<cale_destinatie>]
```

- `x`: dezarhivează (eXtract)
- `v`: afişează ce anume se dezarhivează
- `f nume_arhiva.tar`: numele arhivei care se dezarhivează
- `[<cale_destinatie>]`: opţional, locul unde se realizează dezarhivarea

Observaţie:

(`f` şi `nume_arhiva.tar` se consideră un singur parametru – din acest motiv, de fiecare dată când se foloseşte `f` pentru a indica un fişier, acesta trebuie să apară ultimul în lista de parametrii, fiind urmat imediat de numele fişierului)

Pentru a realiza compresia unui fişier, exista două utilitare mai folosite:

- `gzip`: mai rapid dar cu o rată de compresie mai mică
- `bzip2`: mai lent dar cu o rată de compresie mai mare

Comanda `tar` poate utiliza direct unul dintre programele de compresie menţionate anterior folosind parametrul `z` pentru gzip sau parametrul `j` pentru bzip2, în felul următor:

	Compresie	**Decompresie**
Gzip	`tar czvf fisier.tar.gz dir/`	`tar xzvf fisier.tar.gz`
bzip2	`tar cjvf fisier.tar.bz2 dir/`	`tar xjvf fisier.tar.bz2`

De asemenea, pe lângă opţiunile de compresie/arhivare se mai pot folosi şi alte opţiuni. Printre cele mai utile se numără opţiunea `--preserve`, opţiune ce determină păstrarea drepturilor de acces la arhivare/dezarhivare.

În continuare este prezentat un exemplu de utilizare a comenzii `tar` pentru compresie şi decompresie:

```
[ubuntu@ubuntu:/tmp/q]$ ls
f1.txt  f2.txt  f3.txt
[ubuntu@ubuntu:/tmp/q]$ tar cvf arhiva.tar f*.txt
f1.txt
f2.txt
f3.txt
```

```
[ubuntu@ubuntu:/tmp/q]$ file arhiva.tar
arhiva.tar: POSIX tar archive (GNU)
[ubuntu@ubuntu:/tmp/q]$ gzip arhiva.tar
[ubuntu@ubuntu:/tmp/q]$ ls
arhiva.tar.gz  f1.txt  f2.txt  f3.txt
[ubuntu@ubuntu:/tmp/q]$ file arhiva.tar.gz
arhiva.tar.gz: gzip compressed data, was "arhiva.tar", from Unix, last modified: Thu Sep 20
11:37:30 2007
[ubuntu@ubuntu:/tmp/q]$ ls
arhiva.tar.gz  f1.txt  f2.txt  f3.txt
[ubuntu@ubuntu:/tmp/q]$ mv arhiva.tar.gz fisier.txt
[ubuntu@ubuntu:/tmp/q]$ file fisier.txt
fisier.txt: gzip compressed data, was "arhiva.tar", from Unix, last modified: Thu Sep 20
11:37:30 2007
[ubuntu@ubuntu:/tmp/q]$ ls
f1.txt  f2.txt  f3.txt
[ubuntu@ubuntu:/tmp/q]$ tar czvf new.tar.gz f*
f1.txt
f2.txt
f3.txt
[ubuntu@ubuntu:/tmp/q]$ ls
f1.txt  f2.txt  f3.txt  new.tar.gz
[ubuntu@ubuntu:/tmp/q]$ file new.tar.gz
new.tar.gz: gzip compressed data, from Unix, last modified: Thu Sep 20 11:40:02 2007
[ubuntu@ubuntu:/tmp/q]$ mkdir tmp
[ubuntu@ubuntu:/tmp/q]$ mv new.tar.gz tmp/
[ubuntu@ubuntu:/tmp/q]$ cd tmp/
[ubuntu@ubuntu:/tmp/q/tmp]$ tar xvf new.tar.gz
f1.txt
f2.txt
f3.txt
[ubuntu@ubuntu:/tmp/q/tmp]$ ls
f1.txt  f2.txt  f3.txt  new.tar.gz
```

3.4.8 Backup

Backup-ul este utilizat pentru a păstra într-un loc separat o copie a datelor. Aceasta copie se utilizează pentru a reface datele în cazul în care suportul original nu mai poate fi folosit din orice motiv. Este una dintre cele mai utile acţiuni asupra datelor pe care toată lumea ştie că ar fi bine să o facă dar nu o face. Câteva metode de backup au fost deja prezentate. Tabelul următor prezintă câteva metode de backup şi situaţiile când sunt ele potrivite.

Metoda	**Descriere**
tar + gzip/bzip2	Metodă foarte simplu de aplicat. Devine greu de folosit pentru dimensiuni mari de date. Permite compresia datelor.
dd	Metoda simplu de folosit şi independentă de sistemul de fişiere. Permite păstrarea intactă a structurii sistemului de fişiere. Inflexibilă când vine vorba de recuperarea datelor. Utilă pentru cantităţi mari de date.
rsync	E asemănătoare comenzii cp dar la care s-a adăugat suport de sincronizare intre mai multe computere. Permite replicarea structurii de fişiere (inclusiv permisiuni) între 2 computere.
rdiff-backup	Este un *wrapper* peste rsync. Adaugă suport pentru backup-uri incrementale, adică: la un moment dat se realizează un backup complet pentru un director (asemănător rsync-ului); backup-urile incrementale salvează doar modificările ce s-au făcut de la ultimul backup pana în prezent, indiferent de tipul backup-ului; în acest fel se poate reveni la orice stare anterioara, în măsura în care s-a realizat cel puţin un backup incremental la acea stare.

Mai multe informaţii despre aceste aplicaţii şi modul lor de utilizare se găsesc pe paginile web ale fiecărui proiect.

3.5 Redirectări de comenzi

3.5.1 Descriptorii de fişier

Fiecare program în execuţie utilizează 3 fişiere speciale pentru a interacţiona cu utilizatorul:

- *stdin* – standard input: fişier care reprezintă locul de unde programul îşi citeşte datele de intrare (de obicei tastatura)
- *stdout* - standard output: fişier care reprezintă locul unde programul scrie datele de ieşire (de obicei acesta este consola curenta)
- *stderr* – standard error: fişier care reprezintă locul unde programul scrie mesajele de eroare (de obicei acesta este consola curentă)

În C/C++, există 3 variabile de tipul FILE* cu numele stdin, stdout şi stderr care sunt iniţializate şi disponibile în orice moment. Ele îndeplinesc rolurile descrise anterior şi pot fi folosite ca pe orice alta variabila de tipul FILE*.

Fiecare fişier deschis de o aplicaţie Linux are asociat un indice. Acest indice se numeşte descriptor de fişier. Fişierele descrise anterior au descriptorii de fişier următori:

- stdin – 0
- stdout - 1
- stderr – 2

Orice fişier deschis de aplicaţie va avea un descriptor mai mare sau egal cu 3.

3.5.2 Redirectări

Există situaţii când, pentru o aplicaţie, utilizatorul doreşte să modifice intrarea sau ieşirea – spre exemplu, utilizatorul doreşte ca ieşirea unui program să se facă într-un fişier şi nu în consola curentă, sau intrarea unui program să fie un fişier şi nu tastatura. Acest lucru se poate face doar lucrând la nivelul descriptorilor.

Shell-ul bash permite utilizând o sintaxa speciala execuţia de comenzi cu descriptorii standard modificaţi.

Se întâlnesc următoarele cazuri:

Redirectare		
sursă	**Destinaţie**	**exemplu comandă**
intrare (stdin)	Fişier	`./program <fişier_intrare`
ieşire (stdout)	Fişier	`./program >fişier_ieşire`
eroare (stderr)	Fişier	`./program 2>fişier_erori`
eroare (stderr)	iesire (stdout)	`./program 2>&1`
eroare & iesire	Fişier	`./program 2>&1 >fişier_ieşire_şi_erori`

Se observă că pentru a redirecta ieşirea/intrarea se folosesc semnele "mai mare" pentru iesire (către exteriorul programului) şi "mai mic" pentru intrare (către program). Pentru redirectarea ieşirii de erori se indică descriptorul de fişier „2". Pentru a redirecta către ieşirea standard (stdout), se foloseşte sintaxa "&1".

Pentru redirectarea ieşirii de erori şi iesirii standard către un fişier:

- se redirectează întâi ieşirea de erori către ieşirea standard folosind "`2>&1`"
- se redirectează ieşirea standard către un fişier "`>fişier_iesire_şi_erori`".

Nu se obţine efectul dorit în cazul în care redirectările se fac în ordine inversă.

Tabelul următor prezintă câteva redirectări folosind fişiere speciale:

Comanda	Efect
`./program 2>/dev/null`	mesajele de la ieşirea de erori nu sunt afişate
`./program 2>&1 >/dev/null`	nici un mesaj nu este afişat
`> nume_fişier`	crează un fişier gol cu numele "nume_fişier"
`cat /dev/null > nume_fişier`	crează un fişier cu acelaşi conţinut ca şi /dev/null , adică un fişier gol

3.6 Drepturi de acces

O primă măsură de protecţie a datelor o reprezintă drepturile de acces la fişiere. Atât timp cât un utilizator nu are drepturi de administrator pe un anumit computer, acel utilizator se supune drepturilor de acces la fişiere.

Tabelul următor prezintă drepturi ce pot fi configurate pentru un fişier, fără a particulariza la un anumit tip de sistem de fişiere:

Drept	Descriere
citire	Dreptul de a deschide şi citi conţinutul unui fişier
scriere	Dreptul de a scrie într-un fişier
execuţie	Dreptul de a executa un fişier (aplicaţie) sau, pentru directoare, dreptul de a intra într-un director
modificare	Dreptul de a modifica datele dintr-un fişier existent
ştergere	Dreptul de a sterge un fişier

3.6.1 Utilizatori şi grupuri de utilizatori vs. liste de acces

Există două metode mai întâlnite pentru definirea drepturilor de acces la fişiere:

- drepturi de acces la nivel de utilizator/grup
- liste de acces - Access Lists (ACL)

Prima metoda (drepturi de acces la nivel de utilizator/grup) constă în definirea unor drepturi pentru următoarele entităţi:

- posesorul unui fişier - owner
- grupul din care face parte fişierul (utilizatorii din grupul de care apartine fişierul)
- toţi ceilalţi utilizatori

Un utilizator se poate afla în mai multe grupuri [vezi 2.5.2]. Această metodă este cea mai folosită cale pentru definirea drepturilor de acces. Ea oferă un nivel de protecţie suficient pentru majoritatea situaţiilor.

În cadrul sistemului bazat pe liste de acces, unui fişier i se pot asocia mai mulţi utilizatori şi/sau grupuri de utilizatori. Pentru fiecare dintre aceste entităţi pot fi configurate mai multe tipuri de drepturi. Această variantă poate fi privită ca o extindere a sistemului cu drepturi de acces prezentat anterior. Complexitatea poate face ca sistemul să fie destul de greu de întreţinut.

Fiecare sistem de fişiere oferă un set de drepturi ce pot fi modificate pentru fiecare fişier în parte. Ca diferenţe notabile între sistemele de fişiere se pot aminti următoarele:

FAT32 nu oferă suport pentru drepturi de acces la fişiere - există doar posibilitatea de a marca un fişier ca read-only – orice utilizator poate să schimbe acest drept

majoritatea sistemelor de fişiere în mediile Unix (inclusiv Mac OS) au suport pentru drepturi de acces bazat pe utilizator/grup şi, folosind o extensie, pot fi extinse cu liste de acces.

NTFS are un sistem foarte avansat de drepturi de acces, bazat pe liste de acces – pentru fiecare entitate adăugată în lista unui anumit fişier pot fi configurate mai multe drepturi.

3.6.2 Clasificarea drepturilor de acces

Sistemele de fişiere din mediile Unix care au ca implementare standard drepturile de acces la nivel user/grup au la baza următorul set de drepturi de acces:

- citire (read) = deschidere şi citire de fişiere
- scriere (write) = creare şi scriere de fişiere
- execuţie (execute) = execuţie fişiere/intrare în directoare

În aceste sisteme de fişiere exista 3 entităţi pentru care poate fi stabilită orice combinaţie a drepturilor definite mai sus:

- utilizator (owner) = posesorul fişierului
- grup (group) = grupul de care aparţine fişierul
- altii (others) = utilizatorii care nu fac parte din grupul fişierului şi nici nu sunt posesori

3.6.3 Schimbarea drepturilor de acces

Pentru a vedea drepturile de acces pentru un anumit fişier este suficientă utilizarea comenzii `ls` cu parametrul `-l`. Spre exemplu, pentru a vedea drepturile de acces asupra unui fişier, transmitem ca parametru comenzii `ls` direct numele fişierului:

```
$ ls -l fisier.txt
-rwxr----- 1 ubuntu users 0 2007-09-20 12:47 fisier.txt
```

Drepturile de acces sunt explicate în continuare:

- primul caracter reprezintă tipul fişierului `-` = fisier normal, `d` = director, `p` = pipe, `b` = dispozitiv bloc, `c` = dispozitiv caracter, `l` = legătură etc.
- următoarele 3 caractere (`rwx`) reprezintă drepturile de acces pentru utilizatorul myuser, care este owner-ul fisierului – se observă ca utilizatorul are toate drepturile (scriere/citire/execuţie)
- următoarele 3 caractere (`r--`) reprezintă drepturile de acces pentru utilizatorii care fac parte din grupul „users" (pot să existe mai mulţi utilizatori în acest grup) – se observă că este prezent dreptul de citire dar lipsesc drepturile de scriere şi execuţie.
- următoarele 3 caractere (---) reprezintă drepturile de acces pentru utilizatorii care nu sunt „ubuntu" şi nici nu fac parte din grupul „users" - acest utilizatori nu au nici un drept.

Există două moduri de reprezentare a drepturilor:

- în formă numerică: pentru fiecare entitate exista o cifră în baza 8 care descrie drepturile, câte un bit pentru fiecare drept
- în formă literală: drepturile sunt referite direct prin iniţiala lor, pentru fiecare tip de entitate.

Astfel, pentru exemplul de mai sus, avem drepturile:

	user = u	group = g	others = o
ieşire comanda "`ls`"	`rwx`	`r--`	`---`
în binar	111	100	000
formă numerică	7	4	0
formă literală	`u=rwx`	`g=r`	`o=N/A`

Pentru a schimba drepturile de utilizator asupra unui fişier pot fi folosite comenzi de forma:

```
$ ls -l fisier.txt
-rwxr----- 1 ubuntu users 0 2007-09-20 12:47 fisier.txt
$ chmod 740 fişier.txt
$ ls -l fisier.txt
-rwxr----- 1 ubuntu users 0 2007-09-20 12:47 fisier.txt
```

Comanda anterioară schimbă drepturile fişierului în drepturile pe care le avea înainte (= nu realizează nici o modificare). În cazul formei literale, se pot folosi semnele +, -, = pentru a adaugă, şterge sau seta drepturi. Comanda:

```
$ chmod u-x,g+r,o=r fiier.txt
$ ls -l fisier.txt
-rw-r--r-- 1 ubuntu users 0 2007-09-20 12:47 fisier.txt
```

are ca efect îndepărtarea dreptului de execuţie pentru posesorul fişierului, adăugarea dreptului de citire pentru grupul fişierului şi setarea dreptului de citire pentru alţii.

Drepturile devin astfel:

	user = u	group = g	others = o
ieşire comanda "ls"	rw-	r--	r--
În binar	110	100	100
formă numerică	6	4	4
formă literală	u=rw	g=r	o=r

Pentru a schimba toate drepturile în acelaşi timp folosind forma literală se utilizează caracterul „a" (= all), precum în exemplul următor:

```
$ chmod a=rw-x fisier.txt
$ ls -l fisier.txt
-rw-rw-rw- 1 ubuntu users 0 2007-09-20 12:47 fisier.txt
```

Comanda de mai sus setează pentru toţi utilizatorii drepturile de scriere/citire şi şterge dreptul de execuţie utilizatorilor care îl aveau înainte.

Pe lângă comanda `chmod` mai există şi comanda `chown`, comandă folosită pentru schimbarea owner-ului şi/sau a grupului din care face parte un fişier. Sintaxa aceste comenzi este:

```
chown [<opţiuni>] [<utilizator>]:[<grup>] <fisier>.
```

Utilizatorul sau grupul poate să lipsească din comandă. Pentru a schimba recursiv owner-ul/grupul unui director, utilizează parametrul `-R`.

Comanda de mai jos schimbă owner-ul şi grupul pentru fişierul *file.txt*. în owner=root, grup=users:

```
$ chown root:users file.txt
```

Pentru a schimba doar grupul pentru un director şi directoarele din acesta, se poate utiliza comanda:

```
$ chown -R :users /mnt/music
```

Într-un mod similar, se poate schimba doar owner-ul:

```
$ chown -R uso: /home/uso
```

Mai multe informaţii despre securitatea fişierelor se găsesc în [vezi 8.2.2.3]

3.7 Tendințe în sistemele de fișiere

3.7.1 Sisteme de fișiere în userspace

Sistemele de fișiere pot fi citite/scrise de sistemele de operare doar dacă exista driver-e care să le implementeze. Implementarea unui driver pentru un sistem de fișiere nu este tocmai ușoară. Acesta este și principalul motiv pentru care exista destul de puține sisteme de fișiere care să implementeze un set de funcționalități acceptabil. Majoritatea sistemelor de fișiere prezente în sistemele de operare au mulți an de dezvoltare.

După cum s-a prezentat anterior, există situații în care s-a putut foarte ușor abstractiza accesul la o serie de date prin intermediul fișierelor (precum informații despre sistem, în `/proc`). Practic, aproape orice informație poate fi prezentată sub forma de fișier. Singura problema care a împiedicat mult timp aceasta trecere către reprezentarea de date folosind fișiere a fost dificultatea scrierii unui driver adecvat.

În ultima perioadă a luat amploare utilizarea de driver-e pentru diferite sisteme de fișiere în user-space, spre deosebire de driver-ele clasice care funcționează în kernel-space. Driver-ele din userspace se bazează pe faptul ca exista un driver în kernel care se ocupa de translatarea apelurilor la date între kernel și sistemul de fișiere din userspace. Acest suport pentru sisteme de fișiere în userspace este prezent în kernel-ul Linux cu numele FUSE[12] (*Filesystem în USErspace*) iar Mac OS cu numele MacFUSE[13].

Tabelul următor prezintă avantajele și dezavantajele sistemelor de fișiere în userspace:

Avantaje	Dezavantaje
suport pentru scrierea de driver-e în numeroase limbaje de programare	mai lente decât sistemele de fișiere normale
dezvoltare mult mai rapida datorita atât limbajelor de nivel înalt cât și datorită posibilității de testare mult mai rapidă (+ debugging)	au o flexibilitate mai mică în ceea ce privește nivelul de acces la hardware, comparativ cu un driver de kernel (de obicei nu este o limitare foarte importantă, dat fiind scopul sistemelor de fișiere de acest tip)
nu blochează sistemul în cazul unei proaste funcționari	
compatibilitate a implementărilor între Linux și Mac OS	

Printre cele mai folosite sisteme de fișiere în userspace se numără:

- NTFS-3G – sistem de fișiere care oferă suport read/write pentru NTFS
- sshfs – sistem de fișiere care oferă acces la fișierele de pe altă mașina prin intermediul SSH ca și cum ar fi parte din sistemul de fișiere local
- EncFS – Encrypted Filesystem – sistem de fișiere care criptează automat fișierele din directorul pe care îl folosește ca suport
- AVFS – Anti-Virus Filesystem – sistem de fișiere care realizează scanarea pentru viruși în momentul în care se realizează un acces la fișiere – este implementat ca sistem de fișiere intermediar pentru accesul protejat la alt sistem de fișiere

Fiecare dintre aceste sisteme de fișiere se montează într-un mod diferit, datorită modului în care au fost gândite să lucreze cu datele: unele au nevoie de un nume de utilizator și o parolă, altele au nevoie de calea către un alt sistem de fișiere etc. Mai multe informații se pot găsi pe paginile web ale fiecărui proiect.

---

[12] *FUSE - http://fuse.sourceforge.net/*
[13] *MacFUSE - http://code.google.com/p/macfuse/*

3.7.2 WinFS, ReiserFS4, ext4

WinFS (*Windows Future Storage*) este numele de cod pentru un sistem de fişiere dezvoltat de Microsoft pentru Windows, bazat pe modelul bazelor de date relaţionale. Practic, orice informaţie prezentă în sistemul de fişiere poate fi asociată automat sau manual cu altă informaţie. Aceste relaţii îmbunătăţesc oferă o nouă viziune asupra modului de lucru cu fişierele din sistem. Planul a fost ca *WinFS* să fie inclus în Longhorn (numele de cod pentru versiunea de dezvoltare a Windows Vista) dar a fost retras pe motiv că este încă în dezvoltare.

ReserFS4 este urmaşul lui *ReiserFS3* şi aduce cu sine o serie de îmbunătăţiri precum un sistem de jurnalizare îmbunătăţit, o eficienţă mai mare în lucrul cu fişiere mici, structură flexibilă de plugin-uri (pentru compresie, criptare) etc. *ReiserFS4* nu este încă inclus în kernel-ul Linux dar se găseşte pentru testare în versiunea `-mm` de kernel, întreţinută de Andrew Morton.

Ext4 este un sistem de fişiere jurnalizat dezvoltat ca îmbunătăţire a lui *ext3*. *Ext4* aduce nou suport pentru sisteme de fişiere foarte mari (până la 1 exabyte = 1024 petabytes), extents (posibilitatea de a mapa blocuri mari contigue pentru un singur fişier, compatibilitate înapoi/înainte. Compatibilitatea înapoi este dată de faptul că un sistem *ext3* poate fi montat ca un sistem *ext4*. Compatibilitatea înainte este posibilă deoarece un sistem ext4 poate fi montat ca un sistem *ext3*. *Ext4* a fost inclus în kernel-ul Linux începând cu versiunea 2.6.19 dar la momentul scrierii acestei cărţi el este marcat ca fiind în dezvoltare.

3.8 Studii de caz

3.8.1 Comenzi pentru lucrul cu fişiere în Windows

Comanda Linux	Comanda Windows	Descriere
`comanda --help`	`comanda /?`	afişează informaţii despre comandă
`cd`	`cd`	schimbă directorul curent
`pwd`	`chdir`	afişează directorul curent
`clear`	`cls`	şterge ecranul consolei curente
`cp`	`copy`	copiază un fişier
`rm`	`del`	şterge un fişier
`ls`	`dir`	afişează conţinutul directorului curent
`vim`	`edit`	editează un fişier text
`exit`	`exit`	inchide shell-ul curent
`diff`	`fc`	compară două fişiere şi afişează diferenţele între ele
`find`	`find`	caută fişiere
`mkfs (mke2fs)`	`format`	formatează un disc
`free`	`mem`	afişează informaţii despre memoria liberă şi cea ocupată
`mkdir`	`mkdir`	crează un nou director
`mv`	`move`	mută un fişier
`mv`	`ren`	redenumeşte un fişier
`date`	`time`	afişează ora sistemului
`diff`	`fc`	afişează diferenţele dintre două fişiere

3.8.2 Utilizarea în siguranță a comenzilor pe fișiere

Exista o serie de opțiuni pentru comenzile `cp`, `mv` și `rm` care fac ca utilizarea acestora să fie mult mai sigura din punct de vedere al pierderii datelor în mod accidental:

fiecare comanda are printre opțiuni și `-i`, o opțiune care determină comanda să ceară o confirmare pentru fiecare fișier șters/suprascris. Pentru a anula efectul acestui parametru poate fi folosit parametrul `-f`.

fiecare comanda recunoaște opțiunea `-v`, opțiune care determină afișarea fișierelor procesate la momentul respectiv – poate reprezenta o metodă bună pentru a vizualiza ce anume se întâmplă în spatele comenzii, mai ales dacă se execută comanda pentru un set mare de fișiere.

Aceste două opțiuni (`-iv`) pot fi forțate o dată cu toate execuțiile realizate de utilizator pentru aceste comenzi dacă se realizează alias-uri [vezi 10.3.3]

Alias-urile se descriu în felul următor, într-un fișier care este încărcat la pornirea shell-ului (precum *~/.bashrc*):

```
alias cp="cp -iv"
alias mv="mv -iv"
alias rm="rm -iv"
```

Astfel, la execuția oricărei dintre aceste comenzi, automat se vor lua în considerare și aceste opțiuni.

3.8.3 Montarea unui sistem de fișiere FAT32 astfel încât toți utilizatorii să aibă drept de scriere pe ea

Pentru a realiza acest lucru se utilizează opțiunile de montare a sistemelor de fișierelor. Pentru a forța la montare drepturile fișierelor se utilizează măști (en: masks). Masca este o înșiruire de biți care dacă sunt 1 semnifică un drept care lipsește.

Exista mai multe tipuri de măști:

- umask = mască aplicată tuturor fișierelor noi (default=022 = 010|010|010 = -w-|-w-|-w-: utilizatorii din grupul fișierului și alți utilizatori nu au drept de scriere asupra fișierelor create)
- fmask = mască aplicată tuturor fișierelor (default = umask)
- dmask = mască aplicată tuturor directoarelor (default = fmask)

Folosind aceste măști se definesc optiuni pentru montarea unei partiții FAT32 în felul următor:

```
mount -t vfat -o dmask=000,fmask=111 /dev/sda2 /mnt/media
```

Se observă parametrul `-o dmask=000,fmask=111` prin care se specifică faptul că pentru toți utilizatorii fișierele normale trebuie să apară cu drepturi read/write iar directoarele cu toate drepturile. Trebuie menționat că pe o partiție FAT32 nu ar trebui să se găsească aplicații/scripturi ce pot fi executate din Linux. Pentru a preveni această situație, tuturor fișierelor le este îndepărtat dreptul de execuție, prin masca 111.

Comanda de mai sus se poate transforma foarte ușor într-o linie în `/etc/fstab`:

```
/dev/sda2  /mnt/media   vfat  defaults,dmask=000,fmask=111   0   0
```

S-a păstrat opțiunea `defaults` pentru că ea activează alte opțiuni utile la montarea automată (mai multe detalii se regăsesc în „`man mount`”).

3.8.4 Montarea unui sistem de fişiere FAT32 astfel încât utilizatorii dintr-un anumit grup să aibă drept de scriere pe ea

Exista situaţii când se doreşte ca doar un grup de utilizatori sa aibă acces la o partiţie. În aceste condiţii, posesorul (owner-ul) fişierelor trebuie modificat în root iar grupul în grupul dorit. Grupul trebuie sa aibă toate drepturile. În primul rând trebuie determinat ID-ul grupului care se doreşte a avea drepturi asupra fişierelor. Cunoscându-se numele grupului, comanda următoare va afişa ID-ul grupului.

```
$ cat /etc/group | grep ^numegrup
numegrup:x:100:
```

Comanda filtrează liniile din `/etc/group` şi afişează pe ecran doar liniile care încep cu numele grupului dorit. Numărul ce se găseşte între „:" este ID-ul grupului dorit.

Comanda care montează un sistem de fişiere FAT32 cu restricţiile menţionate mai sus este următoarea:

```
mount -t vfat -o dmask=007,fmask=117,gid=100,uid=0 /dev/sda2 /mnt/media
```

Se observă că, spre deosebire de cazul anterior, au mai fost introduse două opţiuni:

- `gid=100`: ID-ul grupului care va avea drepturi asupra fişierelor
- `uid=0`: ID-ul posesorului fişierelor – root-ul are ID=0

Măştile au fost schimbate pentru a nu permite nici un fel de drept altor utilizatori în afara grupului şi user-ului root (au 7 pe ultima poziţie).

3.8.5 Sistem de fişiere într-un fişier

După cum s-a prezentat şi în subcapitolele anterioare, în Linux se poate accesa (aproape) tot printr-un fişier – în particular, fiecare dispozitiv are asociat un fişier. Spre exemplu, prima partiţie de pe primul HDD are asociat (de obicei) fişierul `/dev/sda1`. În mod similar cu utilizarea fişierului `/dev/sda1`, orice alt fişier poate fi folosit ca suport pentru un sistem de fişiere. Două dintre motivele cele mai folosite pentru a realiza acest lucru sunt:

- experimentarea unor noi sisteme de fişiere, fără a repartiţiona discul
- încapsularea datelor într-un sistem de fişiere (eventual criptat)

În continuare va fi prezentat modul în care se poate realiza un sistem de fişiere stocat într-un fişier. Se vor prezenta în paralel comenzile atât pentru sistemul de fişiere aflat într-o partiţie cât şi pentru sistemul de fişiere aflat într-un fişier.

Acţiune	**Sistem de fişiere**	
	Într-o partiţie	**Într-un fişier**
alocat spaţiu	se partiţionează discul; se obţine partiţia /dev/sda1	`$ dd if=/dev/zero of=~/date bs=1MB count=20` se crează astfel un fişier cu 20 blocuri, fiecare 1MB, se obţine echivalentul unei partiţii de 20MB
formatat	`$ /sbin/mkfs -t vfat /dev/sda1`	`$ /sbin/mkfs -t vfat ~/date`
verificat	-	`$ file ~/date` ~/date: x86 boot sector, mkdosfs boot message display, code offset 0x3c, OEM-ID " mkdosfs", sectors/cluster 4, root entries 512, sectors 39062 (volumes <=32 MB) , [...]
montat	`# mount -t vfat /dev/sda1 dir/`	`# mount -t vfat -o loop ~/date dir/`
demontat	`# umount dir/`	`# umount dir/`

Opțiunea `-o loop` indica sistemului de operare că partiția nu se găseşte pe un dispozitiv fizic ci într-un fişier.

3.8.6 ntfs-3g

nfts-3g este proiectul care oferă suport read-write pentru partiții NTFS sub Linux. Pachetul se găseşte în repository-ul universe.

Acesta trebuie instalat folosind comanda:

```
# apt-get install ntfs-3g
```

Pentru a identifica partiția NTFS se poate utiliza comanda:

```
# fdisk -l | grep NTFS
```

care afişează doar liniile care conțin „NTFS" din ieşirea comenzii `fdisk -l`.

În `/etc/fstab` se adauga o linie de forma:

```
<dispozitiv> <director montare> ntfs-3g defaults,locale=en_US.utf8 0 0
```

şi pentru a monta toate partițiile prezente în `/etc/fstab` se execută comanda:

```
# mount -a
```

Cuvinte cheie

- sistem de fişiere
- montare
- demontare
- fsck
- fişier
- director
- jurnalizare
- cale absolută, cale relativă
- ierarhie
- copiere, mutare, stergere, redenumire
- arhivare
- dezarhivare
- backup
- descriptor de fişier
- redirectare
- drept de acces
- ls, pwd, cd
- cp, mv, rm
- mkdir, rmdir, touch

Întrebări

1 Sistemele de fişiere din prezent oferă un grad mai mare de siguranţă împotriva căderilor prin implementarea:

- ❑ drepturilor de acces la nivel de utilizator şi grup
- ❑ drepturilor de acces prin liste de acces
- ❑ suportului pentru legături hard
- ❑ suportului pentru jurnalizare

2 Care este fişierul folosit pentru montarea automată a unui sistem de fişiere în Linux?

- ❑ /etc/resolv.conf
- ❑ /etc/fsck
- ❑ /mnt/hda5
- ❑ /etc/fstab

3 Un fişier are drepturile 653. Utilizatorul ce deţine fişierul rulează comanda

```
chmod a-x,u+w,g-r,o+r
```

Care sunt noile drepturi ale fişierului?

- ❑ 606
- ❑ 762
- ❑ 534
- ❑ 451

4 Care din următoarele comenzi nu poate fi aplicată asupra unui director?

- ❑ file
- ❑ touch
- ❑ rm -r
- ❑ cat

5 *Care din următoarele nu este o intrare validă în sistemul de fişiere?*

- fişier executabil
- director al cărui nume începe cu .
- legătură simbolică către /bin
- interfață de rețea de looback

6 *Care din următoarele sisteme de fişiere este un sistem nativ Linux?*

- HPFS
- FAT32
- EXT3
- UFS

7 *Un utilizator configurează un fişier cu drepturile 755. Ce utilizator are dreptul de execuție pe fişier?*

- root
- utilizatorul care deține fişierul
- nici un utilizator
- toți utilizatorii

8 *Ce reprezintă litera v din comanda de mai jos?*

```
tar czvf a.tgz dir1
```

- crearea unei arhive
- dezarhivarea unei arhive
- crearea unei arhive GZIP
- afişarea de mesaje despre arhivare

9 *Care comanda este folosită pentru montarea unui sistem de fişiere în Linux?*

- mount
- umount
- mountfs
- fsck

10 *Pentru a putea utiliza un sistem de fişiere pe o partiție dată, prima acțiune absolut necesară care trebuie executată este*

- ştergerea sistemului de fişiere existent anterior
- montarea sistemului de fişiere de pe partiția respectivă
- formatarea partiției cu sistemul de fişiere dorit
- schimbarea drepturilor de acces pentru directorul în care este montată partiția

4 Procese

It's a well-known fact that computing devices such as the abacus were invented thousands of years ago. But it's not well known that the first use of a common computer protocol occurred in the Old Testament. This, of course, was when Moses aborted the Egyptians' process with a control-sea
Tom Galloway

Ce se învață din acest capitol?

- Definiția unui proces; diferența între un program și un proces
- Noțiunea de multitasking
- Ierarhia de procese într-un sistem de operare
- Comenzi de vizualizare a proceselor (`ps`, `pstree`, `top`)
- Daemoni; rularea proceselor în background/foreground
- Noțiunea de semnal; administrarea proceselor prin intermediul semnalelor
- Comunicarea între procese; operatorul | (pipe) de comunicare între procese
- Administrarea proceselor/serviciilor în Windows

4.1 Noțiuni introductive

În lumea sistemelor de operare Unix (şi nu numai) există două abstracții fundamentale: fişierul şi procesul. Fişierul este folosit pentru a abstractiza informația şi modul de stocare şi utilizare a acesteia în sistemul de operare. De partea cealaltă, procesul este folosit pentru a abstractiza executarea sarcinilor sistemului de operare.

4.1.1 Ce este un proces?

În definiția sa cea mai simplă, un proces este un program aflat în execuție. Dacă un program se consideră o entitate pasivă, un proces este considerat e entitate activă.

Procesul este unitatea de bază a sistemului de operare folosită pentru a îndeplini sarcinile indicate de utilizator. Orice solicitare a utilizatorului se translatează în crearea unui proces şi rezolvarea acelei solicitări. Un proces este, aşadar, o entitate de execuție.

4.1.2 Deosebirea dintre un proces şi un program

Un **program** este doar un fişier executabil care se află pe disc. Se mai spune că un fişier executabil sau program este **imaginea unui proces**. Un program este, aşadar, o entitate pasivă, care nu are asociate noțiuni de utilizare a memoriei şi procesorului (elemente de bază într-un sistem de calcul). Fişierele `/bin/ls`, `/usr/bin/vi` sunt exemple de programe:

```
razvan@anaconda:~$ ls -l /bin/ls
-rwxr-xr-x 1 root root 77352 Jan 30 20:51 /bin/ls
razvan@anaconda:~$ ls -l /usr/bin/vi
lrwxrwxrwx 1 root root 20 Oct  6  2005 /usr/bin/vi -> /etc/alternatives/vi
razvan@anaconda:~$ ls -l /etc/alternatives/vi
lrwxrwxrwx 1 root root 18 Apr 22 16:30 /etc/alternatives/vi -> /usr/bin/vim.basic
razvan@anaconda:~$ ls -l /usr/bin/vim.basic
-rwxr-xr-x 1 root root 1383816 Jan 31 20:25 /usr/bin/vim.basic
```

Un program conține, în principal, două componente:

- secțiunea de cod: conține instrucțiunile de rulare a programului pe procesor;
- secțiunea de date: conține informațiile utilizate la execuția programului.

De partea cealaltă, un proces pornește în momentul execuției unui program. În acel moment, sistemul de operare creează o structură (o entitate, o abstracție) denumită proces. Dincolo de informațiile conținute în imagine (program), sistemul de operare asociază procesului o zonă de memorie unde își va menține datele și unde se va afla codul de executat; de asemenea, îi asociază timp de rulare a codului pe procesor și alte informații utile pentru interacțiunea cu alte procese sau componente ale sistemului de operare.

Exemple de procese sunt rularea comenzii `ls` sau rularea editorului `vi`. În momentul rulării comenzii `ls` sistemul de operare folosește imaginea `/bin/ls` și creează un proces. Acest proces este răspunzător pentru utilizarea resurselor sistemului (memorie, procesor) pentru a oferi rezultatul rulării comenzii; în cazul de față acest rezultat este afișarea conținutului directorului curent.

```
razvan@anaconda:~$ ls
Desktop   bin    code          media     packages  public_html  websites
Download  bkup   extra-school  mystuff   people    school
Maildir   books  junk          official  projects  svn-repos
razvan@anaconda:~$ ls -l /bin/ls
-rwxr-xr-x 1 root root 77352 Jan 30 20:51 /bin/ls
razvan@anaconda:~$ file /usr/bin/file
/usr/bin/file: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.4.1,
dynamically linked (uses shared libs), for GNU/Linux 2.4.1, stripped
```

În exemplul de mai sus, rularea comenzii `ls` duce la crearea unui proces din programul (executabilul) `/bin/ls`. Rularea comenzii (și, deci, a procesului) duce la afișarea conținutului directorului curent. A doua comandă are ca argument chiar executabilul folosit pentru crearea procesului. La fel și a treia comandă. Pot fi considerate exemple triviale de *self-reference* (http://en.wikipedia.org/wiki/Indirect_self-reference).

Se observă că un proces este o instanță de execuție a unui program. Putem avea mai multe instanțe pentru același program. Avem, așadar, o asociere *many to one* (mai multe la unul) între procese și programe.

4.1.3 Structura unui proces

Un proces este definit la nivelul sistemului de operare ca o structura de date. Un proces are anumite proprietăți și dispune de anumite resurse. Dintre acestea, cele mai importante sunt:

- **identificatorul unui proces**: acesta numește **PID** (process id). Este vorba de un număr întreg care identifică unic în cadrul sistemului de operare procesul nostru.
- **imaginea procesului**: aceasta este reprezentată de programul din care ia naștere procesul. Imaginea procesului (programul) conține codul și datele necesare pentru execuție.
- **starea procesului**: un proces se poate afla în rulare (în momentul în care codul asociat se execută pe procesor) sau în așteptare (un alt proces se află în rulare).
- **zonele de memorie** utilizate: orice proces utilizează anumite regiuni de memorie pentru a menține codul și datele.
- **fișiere deschise**: un proces menține o tabelă cu fișierele deschise. De obicei dimensiunea tabelei este limitată astfel că și numărul de fișierele pe care le poate

deschide un proces sunt limitate. (În Linux această limită este stabilită implicit la 1024).

4.1.3.1 Vizualizarea structurii unui proces utilizând procfs

În Linux informații despre structura unui proces se pot afla prin inspecția **procfs**. Acesta este un sistem de fișiere virtual care oferă acces la informații din sistemul de operare. Pentru a afla informații despre un anumit proces va trebui accesat directorul `/proc/pid`. `pid` este numărul ce identifică procesul (pid-ul). În exemplul de mai jos a fost accesat directorul procesului cu pid-ul `6740`:

```
razvan@asgard:~$ cd /proc/6740/
razvan@asgard:/proc/6740$ ls
auxv     cwd      exe  maps  mounts      oom_adj    root   stat    status  wchan
cmdline  environ  fd   mem   mountstats  oom_score  smaps  statm  task
```

Diversele intrări din acest director oferă informații legate de proces:

- pid-ul este dat de numele directorului: `6740`
- imaginea procesului este dată de legătura simbolică `exe`:

```
razvan@asgard:/proc/6740$ ls -l exe
lrwxrwxrwx 1 razvan razvan 0 2007-07-19 19:38 exe -> /usr/bin/vim.gnome
```

După cum se vede, procesul de față este o instanță a `vim` în mediu grafic (GNOME).

- linia de comandă utilizată pentru crearea procesului:

```
razvan@asgard:/proc/6740$ cat cmdline
vi 1.txt
```

Se observă că s-a rulat comanda `vi 1.txt`.

- directorul curent - cel în care s-a rulat comanda - este dat de legătura simbolică `cwd`:

```
razvan@asgard:/proc/6740$ ls -l cwd
lrwxrwxrwx 1 razvan razvan 0 2007-07-19 19:38 cwd -> /home/razvan/junk/fd_test
```

Rezultă că directorul curent al comenzii este `/home/razvan/junk/fd_test`.

- zonele de memorie ale procesului sunt date de fișierul `maps`:

```
razvan@asgard:/proc/6740$ cat maps
08048000-081ce000 r-xp 00000000 03:03 180465     /usr/bin/vim.gnome
081ce000-081db000 rw-p 00186000 03:03 180465     /usr/bin/vim.gnome
081db000-08287000 rw-p 081db000 00:00 0          [heap]
[...]
b7fdc000-b7fde000 rw-p 00014000 03:03 178741     /lib/ld-2.3.6.so
bfc7a000-bfc8f000 rw-p bfc7a000 00:00 0          [stack]
```

Nu vom insista pe aceste regiuni întrucât depășesc aria de cuprindere a capitolului de față.

- fișierele deschise de proces pot fi vizualizate prin accesarea subdirectorului `fd`:

```
razvan@asgard:/proc/6740$ cd fd
razvan@asgard:/proc/6740/fd$ ls
0  1  2  3  5  7
razvan@asgard:/proc/6740/fd$ ls -l
total 6
lrwx------ 1 razvan razvan 64 2007-07-19 19:48 0 -> /dev/pts/3
lrwx------ 1 razvan razvan 64 2007-07-19 19:48 1 -> /dev/pts/3
[...]
lrwx------ 1 razvan razvan 64 2007-07-19 19:48 7 -> /home/razvan/junk/fd_test/.1.txt.swp
```

Nu vom insista pe descrierea tuturor intrărilor, dar se poate observa că fișierul cu numărul `7` este o legătură simbolică către `/home/razvan/junk/fd_test/.1.txt.swp`. Acest fișier este copia de lucru a editorului `vim` pentru fișierul `1.txt`.

Sistemul de fișiere procfs este folosit drept suport de comenzile care oferă informații despre procese. Astfel de comenzi sunt `ps`, `pstree`, `pgrep`, `top` despre care vom discuta în secțiunile următoare.

4.1.4 Stările unui proces. Multiprogramare şi multiprocesare

Într-un sistem de operare există, la un moment dat, mai multe procese. Totuşi, la un moment de timp dat, un singur proces poate rula pe un procesor; altfel spus, un procesor poate executa codul asociat unui singur proces. Astfel, pe un sistem cu un singur procesor rulează un singur proces la un moment dat. Pe un sistem cu două procesoare vor putea rula două procese în acelaşi timp. Orice proces are, aşadar, asociată o stare de activitate. În mod obişnuit există mai multe stări în care se poate găsi un proces, dar noi ne vom referi doar la două:

- **rulare** (running): procesul rulează în acel moment pe unul din procesoarele sistemului;
- **aşteptare** (waiting): procesul aşteaptă eliberarea procesorului pentru a putea rula pe acesta.

Se observă, aşadar, că este nevoie de un algoritm care să ofere fiecărui proces acces la unul din procesoarele sistemului.

*Mecanismul de alocare a proceselor pe procesoarele sistemului poartă numele de **planificare** (scheduling). Subsistemul nucleului sistemului de operare care se ocupă de asociere se numeşte **planificator de procese** (scheduler).*

Rolul planificatorului este de a alege un proces care să ruleze pe un procesor şi, la îndeplinirea unei condiţii date, să înlocuiască acel proces cu un altul. Înlocuirea unui proces cu un alt proces se numeşte **schimbare de context** (context switch).

La începutul istoriei sistemelor de operare, dacă un utilizator dorea rularea unui program, el încărca pe un suport conţinutul programului şi aştepta rularea acestuia (în contextul unui proces). Când procesul era planificat pe procesor, acesta rula până la încheierea absolută a execuţiei. Această abordare are dezavantajul lipsei de interactivitate: dacă un proces rulează foarte mult sau dacă trebuie să comunice des cu dispozitive periferice lente, atunci este nevoie de mult timp pentru ca alte procese să ruleze.

Următorul pas a fost introducerea noţiunii de **multiprogramare**. Înainte de a intra în detalii, trebuie amintită diferenţa de viteza între procesor/memorie şi dispozitivele periferice. Astfel, dacă un proces accesează hard disk-ul, va trebui să aştepte foarte mult timp pentru ca informaţia să fie disponibilă.

În abordarea multiprogramării, mai multe procese coexistă în cadrul unui sistem de operare. În momentul în care un proces accesează un dispozitiv periferic lent, va trebui să aştepte furnizarea informaţiei dorite; drept urmare, un alt proces este planificat pe procesor (context switch). În acest fel, se măreşte interactivitatea sistemului. Dacă un proces se blochează în comunicaţia cu un dispozitiv periferic lent, acesta nu mai iroseşte timpul procesorului în aşteptare, ci este înlocuit cu un alt proces.

Următorul pas a fost apariţia conceptului de **multiprocesare** sau **multitasking**. Acesta a venit ca răspuns la cerinţele din ce în ce mai mari de interactivitate a sistemului de operare în raport cu utilizatorul. Problema multiprogramării este că un proces care rulează mult timp pe procesor fără a executa o acţiune blocantă nu va lăsa alte procese să ruleze. Acest lucru înseamnă pierderea interactivităţii. Soluţia este asocierea unui timp de rulare pe procesor, a unei **cuante de timp**. La încheierea acelei cuante de timp, procesul va fi automat înlocuit de un altul. În abordarea multitasking, aşadar, schimbarea de context se va produce când procesul care rulează pe procesor execută o acţiune blocantă sau când îşi încheie cuanta de timp.

NOTĂ: *Toate sistemele de operare moderne sunt sisteme multiprocesare (multitasking).*

4.2 Ierarhia de procese în Unix. Vizualizarea proceselor sistemului

În sistemele Unix (şi, deci, şi în Linux), procesele sunt organizate ierarhic. Astfel, se poate spune despre un proces că este procesul părinte al unui alt proces; viceversa, se spune că un proces este procesul fiu pentru altul. Un lucru de reţinut este faptul că un proces are un singur proces părinte, dar poate avea mai multe procese fiu.

Modelul ierarhiei de procese în Unix este prezentat în figura următoare:

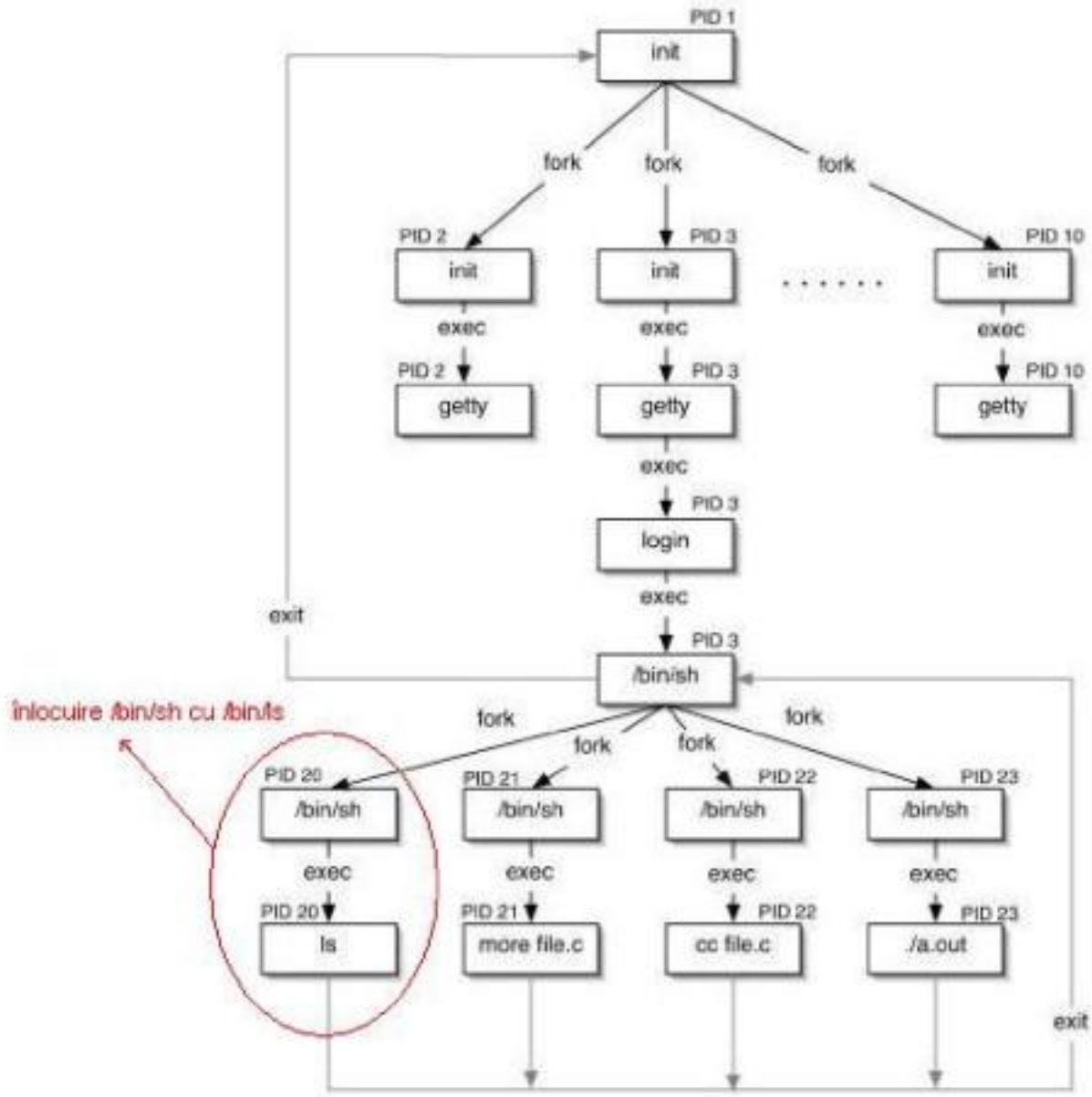


Figura 4-1 Ierarhia de procese în Unix

Ierarhia apare ca urmare a faptului că un proces este creat prin intermediul unui alt proces. La crearea ("naşterea") unui proces se realizează o copie a procesului părinte şi se înlocuieşte imaginea de program cu cea dorită. Astfel, la rularea comenzii `ls` din shell (în imagine `/bin/sh`), se creează o copie a procesului shell, şi se înlocuieşte imaginea de executabil `/bin/sh` cu imaginea `/bin/ls`:

```
razvan@anaconda:~/code$ ls
asm  boot_hello  latex  stack_ovfl  tests
```

În Unix, părintele tuturor proceselor este denumit **init**. Imaginea de executabil asociată este `/sbin/init`. Acesta este primul proces creat de sistemul de operare. Este un proces special pentru că nu are părinte. Din init se creează alte procese, care la rândul lor vor crea alte procese, completându-se astfel ierarhia de procese ale sistemului.

4.2.1 Comanda ps

Comanda `ps` este principala comandă de vizualizare a proceselor sistemului. Informaţiile prezentate de `ps` sunt obţinute din `procfs`. La o rulare simplă (fără argumente), `ps` afişează informaţii despre procesele utilizatorului curent pe terminalul activ:

```
razvan@asgard:~$ ps
  PID TTY          TIME CMD
 3036 pts/0    00:00:00 bash
 3079 pts/0    00:00:00 ps
```

Rularea comenzii presupune afişarea unui cap de tabel şi o intrare pentru fiecare proces. La o rulare simplă se vor afişa identificatorul de proces, terminalul utilizat, timpul de rulare a procesului şi numele acestuia. În exemplul de mai sus, utilizatorul a deschis un terminal cu interpretor de comenzi (`bash`) şi a rulat comanda `ps`. Se observă că un proces `ps` se afişează şi pe sine.

Comanda `ps` acceptă diferite tipuri de opţiuni care pot fi conflictuale uneori. În continuare vom insista pe opţiuni UNIX, caracterizate prin faptul că sunt precedate de semnul minus (-).

Astfel, pentru afişarea proceselor utilizatorului curent, indiferent de terminalul utilizat se foloseşte opţiunea `-a`.

```
razvan@asgard:~$ ps -a
  PID TTY          TIME CMD
 3074 pts/1    00:00:00 vi
 3094 pts/0    00:00:00 ps
```

Se observă că, în această situaţie, se afişează procesele ce rulează pe terminalul `pts/1` şi cele ce rulează pe `pts/0` fără a se mai preciza procesele tip shell de pe aceste terminale.

Comanda `ps` are un număr impresionant de argumente care pot controla conţinutul şi formatul afişării informaţiilor despre procese. Vom prezenta cele două clase importante de opţiuni: opţiuni de selecţie şi opţiuni de formatare.

4.2.1.1 Opţiuni de selecţie

Opţiunile de selecţie presupun selectarea numai a anumitor procese din sistem în funcţie de criteriu. Prezentăm, în continuare câteva opţiuni utile:

- selectarea tuturor proceselor din sistem: se realizează folosind argumentul `-e` sau `-A`

```
razvan@asgard:~$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 ksoftirqd/0
    3 ?        00:00:00 events/0
[...]
 3057 pts/1    00:00:00 bash
 3074 pts/1    00:00:00 vi
 3109 pts/2    00:00:00 bash
 3131 pts/0    00:00:00 ps
```

- selecţia proceselor după o anumită componentă: pid, comandă, terminal, utilizator; această selecţie se realizează prin utilizarea unui argument specific urmat de lista de selecţie (elementele listei sunt separate prin virgula); prezentăm, în continuare, câteva exemple:

```
razvan@asgard:~$ ps -C bash,vi,gnome-terminal
  PID TTY          TIME CMD
 3034 ?        00:00:02 gnome-terminal
 3036 pts/0    00:00:00 bash
 3057 pts/1    00:00:00 bash
 3074 pts/1    00:00:00 vi
 3109 pts/2    00:00:00 bash
 3180 pts/3    00:00:00 bash
```

În exemplul de mai sus, se selectează procesele al căror executabil a fost `bash`, `vi` sau `gnome-terminal`.

```
razvan@asgard:~$ ps -p 1,3026,3074
  PID TTY          TIME CMD
    1 ?        00:00:00 init
 3026 ?        00:00:42 gedit
 3074 pts/1    00:00:00 vi
```

În rularea de mai sus s-au selectat procesele cu pid-ul `1`, `3026` sau `3074`.

```
razvan@asgard:~$ ps -u www-data
  PID TTY          TIME CMD
 2787 ?        00:00:00 apache2
 2788 ?        00:00:00 apache2
 2790 ?        00:00:00 apache2
```

În exemplul anterior, s-au selectat procesele ce aparţineau utilizatorului `www-data`.

Observaţi că deşi s-a făcut selecţia după utilizator, numele acestuia nu apare în rezultatul comenzii. Pentru aceasta vor trebui folosite opţiunile de formatare `ps` pentru afişarea altor câmpuri în afara celor implicite.

Opţiunile pot fi combinate. În exemplul de mai jos se selectează procesele `bash`, `vi`, cele care au pid-ul `2968` sau `2972` sau care aparţin utilizatorului `haldaemon`:

```
razvan@asgard:~$ ps -C bash,vi -p 2968,2972 -u haldaemon
  PID TTY          TIME CMD
 2583 ?        00:00:02 hald
 2590 ?        00:00:00 hald-addon-acpi
 2968 ?        00:00:04 metacity
 2972 ?        00:00:00 nautilus
 3036 pts/0    00:00:00 bash
 3057 pts/1    00:00:00 bash
 3074 pts/1    00:00:00 vi
 3109 pts/2    00:00:00 bash
 3180 pts/3    00:00:00 bash
```

NOTA: argumentele listei de selecţie pentru opţiunile ps sunt separate prin virgulă, ***fără*** spaţii.

Se pot nega opţiunile de selecţie cu ajutorul argumentului `-N`. Astfel, dacă dorim selecţia proceselor care nu aparţin utilizatorului privilegiat (`root`) sau `razvan`, şi care nu au ca executabil `apache2` vom folosi comanda următoare:

```
razvan@asgard:~$ ps -N -u root,razvan -C apache2
  PID TTY          TIME CMD
 2283 ?        00:00:00 portmap
 2575 ?        00:00:00 dbus-daemon
 2583 ?        00:00:03 hald
 2590 ?        00:00:00 hald-addon-acpi
 2655 ?        00:00:00 exim4
 2663 ?        00:00:00 fbguard
 2664 ?        00:00:00 fbserver
 2741 ?        00:00:00 rpc.statd
 2756 ?        00:00:00 atd
```

4.2.1.2 Opțiuni de formatare

Opțiunile de formatare pentru comanda `ps` controlează coloanele care sunt afișate ca rezultat al rulării. Implicit, rularea comenzii oferă un cap de tabel cu elementele `PID`, `TTY`, `TIME`, `CMD`, urmat de o listă a proceselor. Aceste elemente pot fi schimbate prin intermediul opțiunilor de formatare.

- afișarea tuturor opțiunilor frecvente pentru `ps`:

```
razvan@asgard:~$ ps -F
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
razvan    3036  3034  0  1574  3352   0 12:09 pts/0    00:00:00 bash
razvan    3248  3036  0   964   940   0 13:02 pts/0    00:00:00 ps -F
```

Se observă că s-a modificat capul de tabel. Sunt prezentate mai mult opțiuni utile, printre care `UID` (utilizatorul ce a creat procesul), `PPID` (pid-ul procesului părinte), `STIME` (timpul rulării procesului), `CMD` (comanda completă de rulare - cu argumente), etc.

În mod evident, opțiunile de formatare pot fi combinate cu opțiunile de selecție:

```
razvan@asgard:~$ ps -F -C bash,apache2,gnome-terminal,ps
UID        PID  PPID  C    SZ   RSS [...] TTY          TIME CMD
root      2786     1  0  2412  2632 [...] ?        00:00:00 /usr/sbin/apache2 -k start
www-data  2787  2786  0  2355  1948 [...] ?        00:00:00 /usr/sbin/apache2 -k start
www-data  2788  2786  0 57746  2452 [...] ?        00:00:00 /usr/sbin/apache2 -k start
www-data  2790  2786  0 57746  2456 [...] ?        00:00:00 /usr/sbin/apache2 -k start
razvan    3034     1  0  8424 15020 [...] ?        00:00:03 gnome-terminal
razvan    3036  3034  0  1574  3372 [...] pts/0    00:00:00 bash
razvan    3057  3034  0  1575  3348 [...] pts/1    00:00:00 bash
razvan    3109  3034  0  1575  3352 [...] pts/2    00:00:00 bash
razvan    3180  3034  0  1575  3340 [...] pts/3    00:00:00 bash
razvan    3257  3036  0   964   940 [...] pts/0    00:00:00 ps -F -C bash,apache2,gnome-
terminal,ps
```

În exemplul de rulare de mai sus, procesul cu pid-ul `2786` este procesul fiu al init (pid `1`) și că următoarele trei procese sunt procesele fiu ale acestuia. De asemenea, procesul cu pid-ul `3034` (`gnome-terminal`) este procesul fiu al init, iar cele patru procese `bash` sunt procesele fiu ale acestuia. La fel, procesul `ps` (pid `3257`) este procesul fiu al primului proces `bash` (pid `3036`). O reprezentare ierarhică a celor zece procese de mai sus este următoarea:

```
  init (1) ---> apache2 (2786)        ---> apache2 (2787)
                                       \--> apache2 (2788)
                                       \--> apache2 (2790)
           \--> bash-terminal (3034) ---> bash (3036)     ---> ps (3257)
                                       \--> bash (3057)
                                       \--> bash (3109)
                                       \--> bash (3180)
```

Afișarea de opțiuni specifice pentru procese

Specificarea opțoiunilor de afișare pentru `ps` se realizează cu ajutorul argumentului `-o` urmat de argumentele de formatare dorite (separate prin virgulă, fără spații).

Dintre argumentele de selecţie cele mai frecvente amintim: `pid`, `ppid`, `args` (lista de argumente), `user` (utilizator), `comm` (comanda - numai executabil), `cmd` (comanda cu argumente), `cputime` (timpul de procesor consumat), `group`, `pmem` (procent memorie ocupata), `state` (starea procesului - un singur caracter), `stat` (starea procesului - mai multe caractere), `tty` (terminalul de control).

Câteva exemple sunt prezentate în continuare:

- afişarea pid-ului (`pid`), utilizatorului (`user`), stării procesului (`stat`), comanda cu argumente (`cmd`), timpul de procesor (`cputime`) şi memoria folosita (`pmem`):

```
razvan@asgard:~$ ps -e -o pid,user,stat,cmd,cputime,pmem
  PID USER     STAT CMD                              TIME %MEM
    1 root     Ss   init [2]                     00:00:00  0.1
    2 root     SN   [ksoftirqd/0]                 00:00:00  0.0
    3 root     S<   [events/0]                    00:00:00  0.0
[...]
 3009 razvan   S    /usr/lib/gnome-panel/notifi 00:00:00  1.4
 3011 razvan   S    /usr/lib/gnome-applets/mixe 00:00:00  2.3
 3016 razvan   Ss   gnome-screensaver            00:00:02  0.4
[...]
 3180 razvan   Ss+  bash                         00:00:00  0.6
 3260 razvan   R+   ps -e -o pid,user,stat,cmd, 00:00:00  0.1
```

Fără a insista pe semnificaţiile stării proceselor se observă că procesul `ps` (pid `3260`) este în rulare (`R` - running), iar celelalte sunt în aşteptare (`S` - sleeping); sistemul de test dispune de un singur procesor.

- afişarea pid-ului, argumentelor, timpului de procesor, memoriei utilizate şi timpului de pornire a comenzilor aparţinând utilizatorului www-data sau cu numele comenzii bash:

```
razvan@asgard:~$ ps -C bash -u www-data -o pid,args,cputime,rss,start
  PID COMMAND                         TIME   RSS  STARTED
 2787 /usr/sbin/apache2 -k start  00:00:00  1948 11:29:03
 2788 /usr/sbin/apache2 -k start  00:00:00  2452 11:29:03
 2790 /usr/sbin/apache2 -k start  00:00:00  2456 11:29:03
 3036 bash                        00:00:00  3372 12:09:16
 3057 bash                        00:00:00  3348 12:10:20
 3109 bash                        00:00:00  3356 12:23:31
 3180 bash                        00:00:00  3340 12:44:33
```

Spaţiul de memorie utilizat (`rss` - resident set size), este măsurat în kiloocteţi (`KB`).

- sortarea rezultatului rulării comenzii `ps`; se realizează cu ajutorul opţiunii `--sort` urmată de lista de criterii de sortare; semnul c (implicit) sau - în faţa unui criteriu de sortare înseamnă sortare în ordine crescătoare sau descrescătoare:

```
razvan@asgard:~$ ps -e -o pid,cmd,cputime,pmem --sort -rss
  PID CMD                              TIME %MEM
 2740 /usr/bin/X :0 -dpi 96 -audi 00:01:15  4.6
 3026 gedit                       00:01:08  4.3
 2974 gnome-panel --sm-config-pre 00:00:01  3.2
 2972 nautilus --sm-config-prefix 00:00:00  3.2
 3034 gnome-terminal              00:00:04  2.9
[...]
```

În exemplul de mai sus facem o sortare a proceselor din sistem după memoria ocupată. Se observă că procesele cele mai mari consumatoare de memorie sunt sistemul de ferestre (`X`), editorul `gedit` şi elementele mediului grafic GNOME.

4.2.1.3 Alte opțiuni

Utilitarul `ps` permite afișarea ierarhiei de procese în formă arborescentă. Acest lucru se realizează cu ajutorul opțiunii `-H`:

```
razvan@asgard:~$ ps -C init,apache2,gnome-terminal,bash -o pid,user,comm,cputime,pmem --sort pid
-H
  PID USER     COMMAND             TIME %MEM
    1 root     init            00:00:00  0.1
 2786 root       apache2       00:00:00  0.5
 2787 www-data     apache2     00:00:00  0.3
 2788 www-data     apache2     00:00:00  0.4
 2790 www-data     apache2     00:00:00  0.4
 3034 razvan     gnome-termina 00:00:05  2.9
 3036 razvan       bash        00:00:00  0.6
 3057 razvan       bash        00:00:00  0.6
 3109 razvan       bash        00:00:00  0.6
 3180 razvan       bash        00:00:00  0.6
```

Se observă că procesele fiu sunt indentate cu două spatii față de procesul părinte.

4.2.2 Comanda pstree

Comanda `pstree` afișează ierarhia de procese a sistemului. În rularea simplă (fără nici un argument), comanda afișează ierarhia de procese începând de la `init`.

```
razvan@asgard:~$ pstree
init-+-acpid
     |-apache2-+-apache2
     |          `-2*[apache2---26*[{apache2}]]
     |-atd
     |-bonobo-activati
     |-clock-applet
     [...]
     |-gnome-terminal-+-bash
     |                |-bash---vi
     |                |-bash---man---pager
     |                |-bash---pstree
     |                |-gnome-pty-helpe
     |                `-{gnome-terminal}
[...]
```

Dacă dorim afișarea ierarhiei de procese începând de la un anumit proces va trebui să transmitem ca parametru pid-ul acelui proces:

```
razvan@asgard:~$ pstree 3034
gnome-terminal-+-bash
               |-bash---vi
               |-bash---man---pager
               |-bash---pstree
               |-gnome-pty-helpe
               `-{gnome-terminal}
```

Putem afișa inclusiv pid-ul proceselor din ierarhie prin intermediul opțiunii `-p`:

```
razvan@asgard:~$ pstree -p 3034
gnome-terminal(3034)-+-bash(3036)
                     |-bash(3057)---vi(3074)
                     |-bash(3109)---man(3295)---pager(3302)
                     |-bash(3180)---pstree(3343)
                     |-gnome-pty-helpe(3035)
                     `-{gnome-terminal}(3037)
```

Ca și în cazul comenzii `ps`, `pstree` utilizează `procfs` pentru afișarea ierarhiei de procese.

4.2.3 Comanda pgrep

Comanda `pgrep` este echivalentul `grep` pentru lucrul cu procese şi poate fi folosită pentru afişarea selectivă a proceselor din sistem. Funcţionalitatea `pgrep` poate fi îndeplinită cu ajutorul comenzii `ps` şi al argumentelor specifice acesteia: `-p` pentru selecţia după pid, `-C` pentru selecţia după comandă, `-u` pentru selecţia după utilizator, etc.

Sunt prezentate în continuare exemple ale rulării `pgrep` pentru selecţia proceselor care au ca părinte procesul `init` (pid 1). Implicit se selectează doar pid-urile proceselor. Dacă se doreşte şi afişarea unui nume se foloseşte opţiunea `-l`:

```
razvan@asgard:~$ pgrep -P 1
2
3
4
5
1021
2284
[...]
3025
3027
3052
razvan@asgard:~$ pgrep -P 1 -l
2 ksoftirqd/0
3 events/0
4 khelper
5 kthread
1021 udevd
2284 portmap
[...]
3025 gnome-screensav
3027 gnome-terminal
3052 soffice
```

4.2.4 Comanda top

Comanda `top` este utilizată pentru a afişa informaţii în mod dinamic despre procesele existente în sistem. O utilizare frecventă a acesteia este monitorizarea sistemului [vezi 9.3.1.1]

La o rulare fără argumente, `top` va afişa un ecran cu informaţii despre sistem şi procesele care rulează. În mod implicit, acest ecran este actualizat la fiecare `3` secunde. Un exemplu de ecran afişat prin rularea comenzii `top` este prezentat mai jos:

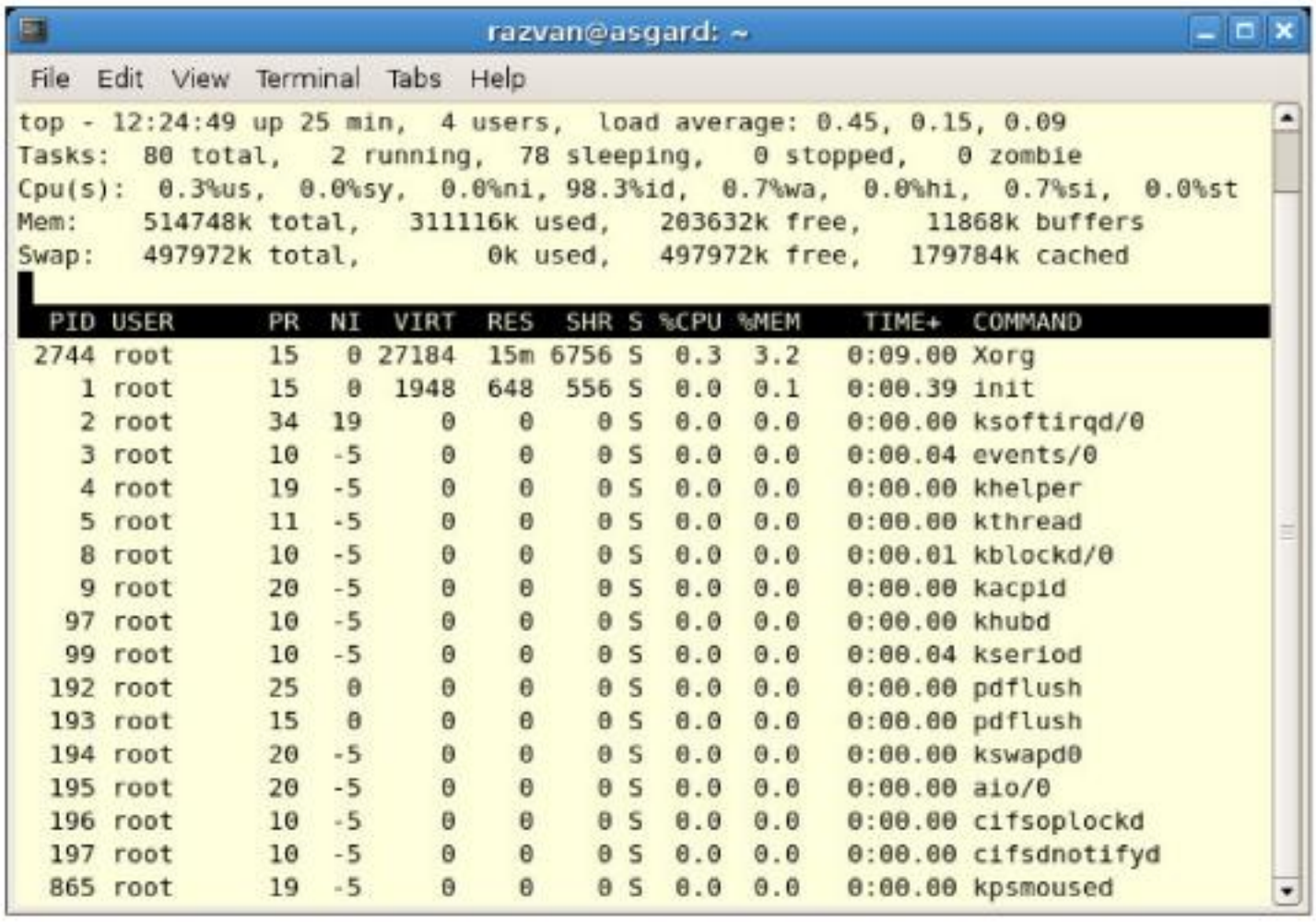


Figura 4-2 Ecran de rulare top

Ecranul afişat de `top` cuprinde următoarele zone:

- **zona de sumarizare** (*summary area*): aceasta conține informații generale despre sistem: încărcarea sistemului (prima linie), numărul de procese şi starea lor (a doua linie), timpul de procesor utilizat (a treia linie), memoria şi spațiul de swap ocupat (ultimele două linii); gradul de încărcare a sistemului este o măsura a utilizării procesorului; poate fi determinat şi cu ajutorul comenzii `uptime`; pentru mai multe detalii despre timpul de procesor utilizat se poate consulta secțiunea [vezi 4.2.5]; detalii despre swap se găsesc în secțiunea [vezi 4.6];
- **zona de comandă** (prompt line): aici utilizatorul poate introduce comenzi specifice de interacțiune cu utilitarul `top`;
- **antetul de tabel** (columns header): afişează antetul de tabel pentru informațiile afişate despre procese;
- **zona de procese** (task area): afişează informații despre procese.

4.2.4.1 Interfața interactivă top

`top` are o interfața interactivă cu utilizatorul care îi permite acestuia alterarea formatului de afişare. Comanda cea mai utilă pentru lucrul interactiv cu top este acea de afişare a ecranului de ajutor, cu ajutorul tastei `h`.

Majoritatea comenzilor `top` sunt de tipul *toggle* adică afişarea sau oprirea afişării unei anumite componente din ecranul de afişare. Vom prezenta câteva din comenzile cele mai utile top în funcție de zona ecranului de afişare în care au efect. Detalii suplimentare se pot afla prin consultarea paginii de ajutor (prin apăsarea tastei `h`) sau a paginii de manual (`man top`).

Comenzi pentru zona de sumarizare

Comenzile folosite pentru lucrul cu zona de sumarizare sunt comenzi de tipul *togle*:

- tasta `l` este folosită pentru a activa/dezactiva afişarea de informaţii despre încărcarea sistemului (primele două linii);
- tasta `t` este folosită pentru a activa/dezactiva afişarea de informaţii despre numărul de procese şi utilizarea procesorului;
- tasta m este folosită pentru a activa/dezactiva afişarea de informaţii despre utilizarea memoriei sistemului.

Comenzi pentru zona de procese

Dintre comenzile care afectează zona de afişare pentru procese amintim:

- tasta `c` este folosită pentru a afişa comanda completă, nu doar numele procesului;
- tasta `f` este folosită pentru a adăuga/elimina coloane utilizate pentru afişarea de informaţii despre procese; tasta `o` este folosită pentru a schimba ordinea criteriilor în capul de tabel; după apăsarea acestor taste, utilizatorului îi este prezentat un ecran de configurare, în care poate specifica noi coloane sau o nouă ordine (tot prin intermediul unor taste); tasta `R` (tip *toogle*) este folosită pentru sortare ascendentă sau descendentă;
- tasta `F` sau tasta `O` este folosită pentru a configura coloana după care se face sortarea proceselor din zona de procese; în mod implicit sortarea se realizează după timpul de procesor; astfel procesele care într-un interval dat de timp au utilizat cel mai mult procesoroul vor apărea primele; tastele `<` şi `>` pot fi folosite pentru schimbarea coloanei de sortare prin parcurgerea criteriilor posibile către stânga sau dreapta.

Comenzi în zona de comandă

Aceste comenzi folosesc zona de comandă şi afectează, de obicei, procesele sau zona de procese:

- tasta `k` permite terminarea unui proces; apăsarea acestei taste oferă promptul `PID to kill:`; aici utilizatorul va introduce pid-ul procesului a cărui execuţie se doreşte a fi încheiată;
- tasta `n` permite precizarea numărului de procese care vor fi afişate în zona de procese; apăsarea acestei taste conduce la apariţia unui prompt unde utilizatorul precizează numărul de procese pe care doreşte să le vizualizeze/monitorizeze;
- tasta `u` permite precizarea unui utilizator; după aceasta, vor fi afişate numai procesele care aparţin acestuia; zona de comandă afişează promptul `Which user (blank for all):` unde se cere introducerea unui nume de utilizator.

4.2.4.2 Interfaţa neinteractivă top

`top` poate fi utilizat şi în mod neinteractiv prin folosirea argumentului `-b` în linie de comandă. Folosit de obicei, împreună cu argumentul `-n` de precizarea a numărului de iteraţii de afişare, se obţine un rezultat ce poate fi redirectat într-un fişier pentru analiză ulterioară:

```
razvan@asgard:~$ top -b -n 1
top - 13:10:00 up  1:10,  4 users,  load average: 0.07, 0.02, 0.01
Tasks:  83 total,   1 running,  82 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.0%us,  0.4%sy,  0.0%ni, 95.8%id,  1.5%wa,  0.1%hi,  0.3%si,  0.0%st
Mem:    514748k total,   319728k used,   195020k free,    15620k buffers
Swap:   497972k total,        0k used,   497972k free,   180864k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      15   0  1948  648  556 S  0.0  0.1   0:00.39 init
    2 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    3 root      10  -5     0    0    0 S  0.0  0.0   0:00.10 events/0
    4 root      19  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
    5 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
    8 root      13  -5     0    0    0 S  0.0  0.0   0:00.01 kblockd/0
[...]
```

4.2.5 Timpul de execuție al unui proces. Comanda time

Timpul de execuție a unui proces se referă la durata de execuție a acestuia în sistem, din momentul creării şi până în momentul terminării acestuia.

Comanda `time` este utilizată pentru măsurarea timpului de rulare a unui proces şi a resurselor utilizate de acesta. Rezultatul rulării implicite a comenzii `time` oferă 3 timpi:

real – timpul efectiv de execuție, măsurat de la crearea procesului şi până la terminarea acestuia;

user – timpul petrecut în spațiul utilizator;

sys – timpul petrecut în sistem (spațiul kernel).

Un exemplu de rulare este prezentat mai jos:

```
asgard:/home/razvan/projects/it2/barc/pictures# time updatedb

real    4m19.309s
user    0m2.680s
sys     0m5.550s
```

Se observă că rularea comenzii `updatedb` a durat 4 minute şi 19 secunde. Totuşi timpul real de rulare a procesului se măsoară adunând timpul petrecut în spațiul utilizator (2.68 secunde) şi timpul petrecut în spațiul kernel (5.55 secunde). Cu alte cuvinte, din timpul cât a durat rularea comenzii, procesul a consumat doar 8.23 secunde. Restul timpul a fost ocupat cu rularea altor procese sau schimbarea contextului între procese. Schimbarea contextului are loc în momentul în care procesului îi expiră cuanta de timp sau accesează dispozitive periferice lente.

4.2.6 Sistemul de fişiere procfs

După cum s-a prezentat, sistemul de fişiere `procfs` este utilizat de comenzile ce obțin informații despre procesele din sistem: `ps`, `pstree`, `psgrep`, `top`. Pachetul Debian/Ubuntu utilizat pentru instalarea acestor utilitare se numeşte `procps` pentru a indica faptul că `procfs` este utilizat pentru obținerea de informații despre procesele sistemului.

În capitolul s-a precizat că `procfs` este un sistem de fişiere virtual (adică nu există un suport fizic precum un hard disk sau CD-ROM). În cazul `procfs`, fiecare fişier este de fapt o regiune de memorie care oferă informații despre sistem. `procfs` oferă informații atât despre procesele din sistem cât şi alte informații specifice: dispozitivele existente, ocuparea întreruperilor permițând totodată şi configurarea anumitor componente ale sistemului.

`procfs` este montat în `/proc`:

```
razvan@asgard:~$ cat /etc/mtab
/dev/hda3 / ext3 rw,errors=remount-ro 0 0
tmpfs /lib/init/rw tmpfs rw,nosuid,mode=0755 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
[...]
```

`procfs` este vizibil utilizatorului ca o suită de fişiere şi directoare cu informații despre sistem:

```
razvan@asgard:~$ cd /proc/
razvan@asgard:/proc$ ls -F
1/     2584/  2792/  2961/  3027/  97/          filesystems  mtrr
1021/  2585/  2793/  2963/  3028/  99/          fs/          net/
192/   2591/  2794/  2969/  3029/  acpi/        ide/         partitions
193/   2603/  2796/  2971/  3052/  asound/      interrupts   self@
194/   2656/  2873/  2976/  3066/  buddyinfo    iomem        slabinfo
```

4.2.6.1 Informații despre procese

Informații despre procesele din sistem sunt oferite de `proc` prin intermediul directoarelor al căror nume este un număr. Acest număr este, de fapt pid-ul procesului despre care se dorește aflarea de informații.

În continuare vom prezenta diverse informații conținute în directorul din `/proc` asociat unui proces. Vom prezenta ca studiu de caz chiar procesul folosit pentru editarea documentului din care a rezultat acest capitol :-). Fiind vorba de OpenOffice, numele executabilului din care a fost generat procesul este `soffice.bin`:

```
razvan@asgard:/proc$ ps -C soffice.bin
 PID TTY          TIME CMD
 3066 ?        00:01:02 soffice.bin
razvan@asgard:/proc$ cd 3066/
razvan@asgard:/proc/3066$ ls -F
auxv     environ  maps    mountstats  root@  statm   wchan
cmdline  exe@     mem     oom_adj     smaps  status
cwd@     fd/      mounts  oom_score   stat   task/
```

Se observă că procesul asociat are pid-ul `3066` și vom accesa, așadar directorul `3066`. Acest director conține o serie de fișiere, directoare și legături simbolice care oferă informații despre proces. Astfel, legătura simbolică `exe`o oferă informații despre executabilul utilizat pentru crearea procesului; fișierul `cmdline` și fișierul `environ` precizează linia de comandă folosită și mediul de creare (environment):

```
razvan@asgard:/proc/3066$ ls -l exe
lrwxrwxrwx 1 razvan razvan 0 2007-08-04 12:01 exe -> /usr/lib/openoffice/program/soffice.bin
razvan@asgard:/proc/3066$ cat cmdline
/usr/lib/openoffice/program/soffice.bin-writer-splash-pipe=5
razvan@asgard:/proc/3066$ cat environ
SSH_AGENT_PID=2950SHELL=/bin/bashGTK_RC_FILES=/etc/gtk/gtkrc:/home/razvan/.gtkrc-1.2-
gnome2OLDPWD=/usr/lib/openofficeUSER=razvanOPENOFFICE_MOZILLA_FIVE_HOME=/usr/lib/openoffice/prog
ramLD_LIBRARY_PATH=/usr/lib/openoffice/program
[...]
```

Fișierul `status` oferă, printre altele, informații referitoare la utilizatorul ce deține fișierul (`uid 1000`), starea procesului (`S` – sleep), zonele de memorie ocupate, numărul de thread-uri:

```
razvan@asgard:/proc/3066$ cat status
Name:   soffice.bin
State:  S (sleeping)
SleepAVG:       98%
[...]
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
[...]
VmPeak:   205720 kB
VmSize:   202392 kB
VmLck:         0 kB
VmHWM:     84040 kB
VmRSS:     80968 kB
[...]
Threads:        6
[...]
```

Directorul `task/` oferă informații despre thread-urile acestui proces. Nu insistăm pe noțiunea de thread, deoarece depășește aria de cuprindere a acestui capitol.

```
razvan@asgard:/proc/3066$ cd task/
razvan@asgard:/proc/3066/task$ ls
3066  3067  3068  3069  3070  3071
```

Directorul `fd/` deține legături simbolice către fișiere deschise de procesul curent. Se observă, printre fișiere deschise, și fișierul în care se editează acest capitol (`04_Procese.doc`):

```
razvan@asgard:/proc/3066$ cd fd/
razvan@asgard:/proc/3066/fd$ ls
0  10  12  14  16  18  2   22  24  26  28  3   31  33  35  37  4  6  8
1  11  13  15  17  19  20  23  25  27  29  30  32  34  36  39  5  7  9
razvan@asgard:/proc/3066/fd$ ls -l
total 38
lr-x------ 1 razvan razvan 64 2007-08-04 13:31 0 -> /dev/null
l-wx------ 1 razvan razvan 64 2007-08-04 13:31 1 -> pipe:[8770]
l-wx------ 1 razvan razvan 64 2007-08-04 13:31 10 -> pipe:[9795]
[...]
lr-x------ 1 razvan razvan 64 2007-08-04 13:31 22 -> /dev/urandom
[...]
lrwx------ 1 razvan razvan 64 2007-08-04 13:31 34 -> /home/razvan/projects/carte-uso/04 -
Procese/04_Procese.doc
[...]
```

4.2.6.2 Alte informații utile

În afara informațiilor despre procese, `procfs` oferă şi alte informații utile despre sistem. Acestea pot fi aflate prin accesul altor fişiere directoare din `/proc`. Printre informațiile utile (folosite şi de alte utilitare din sistem), amintim:

- aflarea informațiilor despre procesor: prin accesarea `/proc/cpuinfo`
- aflarea informațiilor despre memoria sistemului: prin accesarea `/proc/meminfo`
- aflarea informațiilor despre partiții: prin accesarea `/proc/partitions`
- aflarea timpului de rulare a sistemului: prin accesarea `/proc/uptime`

În plus, accesarea directoarelor de forma `fs/`, `net/`, oferă informații ulterioare, permițând, totodată configurarea unor parametri ai sistemului.

4.3 Rularea proceselor în background. Job-uri. Daemoni

Rularea unui proces din interpretorul de comenzi (shell) rezultă, de obicei, în afişarea unui rezultat şi terminarea procesului. Spre exemplu, dacă se doreşte afişarea conținutului directorului curent se rulează comanda `ls`. Acest lucru înseamnă crearea unui proces fiu din shell, încărcarea executabilului `/bin/ls` şi execuția noului proces încheiată cu terminarea procesului. După terminarea procesului, utilizatorul poate rula un nou proces prin introducerea comenzii corespunzătoare la promptul interpretorului.

Un proces care rulează conform scenariului de mai sus se spune că rulează în **foreground** (în prim plan). Un proces care rulează în foreground are acces la terminalul curent. Altfel spus, procesele care rulează în foreground pot citi de la intrarea standard (standard input) şi pot afişa informații la ieşirea standard (standard output).

De partea cealaltă, un proces poate rula în background (în fundal). Un proces care rulează în fundal pierde posibilitatea de a citi de la intrarea standard, dar îşi continuă rularea.

4.3.1 Rularea unui proces în background

Rularea unui proces în background se realizează cu ajutorul operatorului & după comandă. Acest operator va crea procesul asociat comenzii introduse şi va forța rularea acestuia în background. Avantajul acestei abordări este faptul că se oferă promptul interpretorului către utilizator; în felul acesta, utilizatorul poate introduce o nouă comandă. De obicei acest lucru se realizează când comanda de executat durează mult timp: căutarea unui fişier, actualizarea bazei de date de căutare (`updatedb`), rularea unui proces cu interfață grafică, etc. Procesele care rulează în background sunt denumite job-uri.

În exemplul de mai jos, se rulează comenzile de deschidere a editorului `XEmacs` şi a calculatorului din mediul grafic GNOME:

```
razvan@asgard:~$ xemacs &
[1] 3418
razvan@asgard:~$ gnome-calculator &
[2] 3420
razvan@asgard:~$
```

După rularea unei comenzi în background se observă că apare un mesaj specific de forma `[N] M`. `N` este indicele job-ului (indicele procesului care rulează în background), iar `M` este identificatorul de proces (pid-ul) pentru job-ul din background. Avantajul rulării acestor procese în background este faptul că interpretorul de comenzi oferă promptul utilizatorului permițându-i, astfel, acestuia rularea de noi comenzi.

4.3.2 Suspendarea unui proces

Un proces care rulează în foreground poate fi forțat în background în stare suspendat (nu rulează) prin intermediul combinației de taste `CTRL-Z` (`Control` ținut apăsat, după care se apasă `z`). Apăsarea acestei combinații de taste forțează procesul ce rulează în momentul de față în foreground să fie suspendat.

În exemplul de mai jos, utilizatorul a dorit editarea fișierului `out.txt` folosind `vi`. Procesul creat prin execuția comenzii `vi out.txt` rulează în foregroud. Utilizatorul a apăsat `CTRL-Z` și a forțat suspendarea procesului curent în background și oferirea promptului către utilizator.

```
razvan@asgard:~$ vi out.txt
[3]+  Stopped                 vi out.txt
razvan@asgard:~$
```

Se observă că procesul a fost suspendat (stopped) și s-a creat jobul cu numărul `3`.

La fel și în exemplul de mai jos:

```
razvan@asgard:~$ du -hs *
1022M   Desktop
12K     Download
528K    bin
24M     books
756K    code
532K    junk
3.2M    official
4.0K    out.txt
29M     packages
165M    people
617M    pictures
116M    projects

[4]+  Stopped                 du -hs *
razvan@asgard:~$
```

4.3.3 Controlul job-urilor

Ne-am referit prin job la procesele care rulează sau sunt suspendate în background. Interpretorul de comenzi (shell-ul) bash pune la dispoziția utilizatorului mai multe comenzi prin intermediul cărora se poate interacționa cu procesele din background. Mai multe informații despre aceste comenzi pot fi găsite prin consultarea paginii de manual (`man bash`, secțiunea `JOB CONTROL`) sau pagina info (`info bash "Job control"`; trebuie instalat pachetul `bash-doc`).

4.3.3.1 Comanda jobs

Comanda `jobs` afişează job-urile pentru terminalul curent:

```
razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]   Running                 gnome-calculator &
[3]-  Stopped                 vi out.txt
[4]+  Stopped                 du -hs *
razvan@asgard:~$
```

Se observă că avem patru job-uri dintre care două sunt în rulare (`Running`) iar celelalte două sunt suspendate (`Stopped`). Semnul `+` este folosit pentru job-ul curent; acesta este cel mai recent job. Semnul `-` este pentru job-ul anterior job-ului curent.

4.3.3.2 Comenzile bg şi fg

Utilizatorul poate determina rularea comenzilor suspendate. Procesele asociate pot rula în foreground sau în background. Daca se doreşte rularea unui job suspendat în background se foloseşte comanda `bg`; daca se doreşte rularea unui job suspendat în foreground se foloseşte comanda `fg`.

În exemplul de mai jos, s-a decis rularea job-ului `[4]` în background. Rularea comenzii `bg` fără nici un parametru înseamnă continuarea job-ului curent. Daca se doreşte rularea job-ului n se foloseşte argumentul `%n` (spre exemplu `bg %5` pentru rularea în background a job-ului cu indicele `5`).

```
razvan@asgard:~$ bg %4
[4]+ du -hs * &
1.2G    school
2.5M    shared-projects
1.7G    vmware
razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]   Running                 gnome-calculator &
[3]+  Stopped                 vi out.txt
[4]-  Done                    du -hs *
razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]-  Running                 gnome-calculator &
[3]+  Stopped
```

Se observă că s-a continuat rularea job-ului `[4]` în background după care procesul asociat s-a terminat.

Mesajul `Done` este afişat la încheierea rulării unor job-uri. Spre exemplu, daca vom închide din mediul grafic `gnome-calculator` şi `xemacs`, mesajele afişate vor fi asemănătoare cu cele de mai jos:

```
razvan@asgard:~$
[1]   Done                    xemacs
[2]-  Done                    gnome-calculator
razvan@asgard:~$ jobs
[3]+  Stopped                 vi out.txt
```

După terminarea celor două job-uri, mai rămâne în background doar job-ul `3` (cel asociat editorului `vi`). Acesta poate fi adus în foreground utilizând comanda `fg` sau `fg %3`:

```
razvan@asgard:~$ fg
vi out.txt
```

După rularea acestei comenzi, utilizatorul poate continua editarea fişierului `out.txt`.

4.3.4 Daemoni

Un **daemon** este un tip particular de proces care rulează în background. Față de procesele care rulează în background (prin intermediul operatorului `&`) sau care sunt suspendate în background (prin intermediul combinației `CTRL-Z`), un daemon se detașează de terminalul de control. Astfel, un daemon nu va putea comunica direct cu utilizatorul prin intermediul terminalului: nu va accepta comenzi de control de la tastatură şi nu va afişa rezultate la terminal.

Inițial denumirea de daemon nu a avut nici o semnificație. Mai târziu, aceasta a fost abreviată (backronym) la Disk And Execution MONitor.

De obicei, procesele care sunt daemoni au numele terminat în `d`: `hald`, `udevd`, `sshd`, `inetd`, etc. (fără a fi însă o regulă). Pentru selectarea proceselor care nu au ataşate nici un terminal, se foloseşte `ps` cu argumentul `-t` (selecție după terminal) urmat de opțiunea `-` (semnul minus înseamnă procesele care nu au ataşate nici un terminal).

```
razvan@asgard:~$ ps -t -
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 ksoftirqd/0
    3 ?        00:00:00 events/0
[..]
 2657 ?        00:00:00 exim4
 2665 ?        00:00:00 fbguard
 2666 ?        00:00:00 fbserver
 2676 ?        00:00:00 inetd
 2689 ?        00:00:00 sshd
 2695 ?        00:00:00 vsftpd
[...]
 3006 ?        00:00:02 gedit
 3009 ?        00:00:00 gnome-screensav
 3016 ?        00:00:00 gnome-terminal
 3017 ?        00:00:00 gnome-pty-helper
```

Se observă că nu toate procesele care nu au ataşate un terminal sunt procese daemon. Spre exemplu, procesul `gedit (3004)` nu este un daemon în adevăratul sens al cuvântului. Deşi nu are ataşat nici un terminal de control, se poate comunica prin intermediul mediului grafic. În plus față de a nu avea asociat nici un terminal, un proces daemon va rula în background, eliminând posibilitatea comunicării directe (tastatura/monitor) cu utilizatorul.

4.3.4.1 Interacțiunea cu procesele daemon

Se pune însă întrebarea cum poate utilizatorul să interacționeze cu procesele daemon, în măsura în care nu se poate prin intermediul intrării şi ieşirii standard. Cele două mecanisme care permit acest lucru sunt fişierul/fişierele de configurare şi semnalele.

În general, procesele daemon au unul sau mai multe fişiere de configurare. De exemplu:

- server-ul web **apache2** are fişierul de configurare `/etc/apache2/apache2.conf`
- server-ul SSH **openssh** are fişierul de configurare `/etc/ssh/sshd_config`
- server-ul **X** (sistemul de gestiune a mediului grafic) are fişierele de configurare în `/etc/Xorg/`

Aceste fişiere de configurare sunt citite în momentul creării procesului daemon şi permit controlul comportamentului acestuia. În cazul server-ului openssh, acesta permite controlul tipului de autentificare (parolă, chei publice), portul pe care acesta ascultă, dacă permite X forwarding, etc. [vezi 8.2.2].

Dacă se doreşte alterarea comportamentului server-ului, se editează fişierul de configurare. După repornirea procesului daemon, acesta va citi fişierul de configurare şi va stabili noul comportament conform configurației din acesta.

Oprirea, pornirea, repornirea unui proces, în general, şi a unui daemon, în particular, se realizează cu ajutorul semnalelor. Daca în cazul unui proces care nu este daemon, acesta poate fi oprit sau suspendat şi prin intermediul terminalului (combinaţii de taste), un proces daemon poate fi oprit/suspendat/repornit doar cu ajutorul semnalelor. Despre semnale se va discuta în secţiunea 4.4.

4.3.4.2 Comanda nohup

Comanda `nohup` poate fi utilizată pentru a rula un proces cu caracteristici de daemon, adică să forţeze rularea acestuia în background fără interacţiune din partea semnalului de repornire (SIGHUP). `nohup` este folosit, de obicei, atunci când utilizatorul va dori părăsirea terminalului dar comanda va trebui să continue. Comanda `nohup` este urmată de comanda necesară pentru rularea procesului şi de caracterul `&`. Un exemplu este prezentat în continuare:

```
asgard:/home/razvan# nohup updatedb &
[1] 3116
asgard:/home/razvan# nohup: appending output to `nohup.out'
asgard:/home/razvan# ps -e
[...]
 3116 pts/1    00:00:00 updatedb
 3124 pts/1    00:00:00 updatedb
 3125 pts/1    00:00:00 find
 3126 pts/1    00:00:00 sort
 3127 pts/1    00:00:00 frcode
 3129 pts/1    00:00:00 ps
```

Se observă că procesul creat (`updatedb`) nu pierde controlul terminalului curent, însă nu se poate comunica prin intermediul semnalelor de suspendare/repornire sau intrării/ieşirii standard. După cum reiese şi din rezultatul rulării comenzii, ieşirea acesteia este redirecţionată în fişierul `nohup.out`.

4.4 Semnale

Semnalele *sunt mecanisme de notificare asincronă care sunt utilizate pentru a transmite o condiţie specială unui proces.*

Semnalele sunt asincrone în raport cu fluxul de rulare al procesului; astfel, un semnal nu este corelat direct cu instrucţiunile executate de proces; semnalul poate fi transmis la un moment de timp necorelat cu codul executat de proces.

Semnalele sunt transmise în două moduri:

- **de nucleu** (kernel) pentru a indica o condiţie neobişnuită care solicită oprirea/suspendarea sau doar notificarea procesului;
- **de utilizator**.

Semnale care sunt transmise uzual de nucleu sunt:

- `SIGSEGV` (signal segment violation): în momentul accesării invalide a unei zone de memorie;
- `SIGBUS`: în momentul unei erori pe magistrala sistemului;
- `SIGFPE` (signal floating point error): în momentul apariţiei unei erori de virgulă mobilă;
- `SIGTERM` (signal termination): pentru oprirea unui proces;
- `SIGKILL` (signal kill): pentru oprirea necondiţionată a unui proces.

Semnale care sunt transmise uzual de utilizator sunt:

- `SIGINT` (signal interrupt): întrerupe procesul curent;
- `SIGQUIT` (signal quit): semnalează oprirea procesului curent;

- `SIGSTOP` (signal stop): suspenda procesul curent;
- `SIGCONT` (signal continue): reporneşte procesul suspendat.

Nu vom insista pe modul şi condiţiile în care nucleul transmite semnale unui proces, ci modul în care utilizatorul le transmite. Utilizatorul transmite semnale în două moduri unui proces:

- prin intermediul unor comenzi specifice: `kill`, `killall`; [vezi4.4.2]
- prin intermediul unor combinaţii de taste specifice. [vezi 4.4.3]

NOTĂ: *informaţii importante despre semnale se pot afla prin consultarea paginii de manual (man 7 signal)*

4.4.1 Semnale importante UNIX

Înainte de a discuta despre modul de interacţiune a utilizatorului cu procesele prin intermediul semnalelor, se cuvine să amintim câteva semnale importante (unele au fost amintite şi anterior). Lista cu semnale a sistemului poate fi vizualizată folosind comanda `kill` cu opţiunea `-l` (list):

```
razvan@asgard:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Se observă că un semnal are un număr de identificare şi un nume care oferă indicaţii despre efectul său asupra proceselor. Semnalele relevante sunt cele până la `31` (`SIGSYS`). Celelalte (`SIGRTMIN`, `SIGRTMAX`) sunt semnale de timp real (`SIGnal Real Time`) şi nu sunt relevante pentru prezentarea curentă. Vom prezenta câteva dintre semnalele cele mai importante:

- `SIGHUP` (`1`) (signal hang up): este folosit pentru repornirea unui proces; acest semnal este de obicei transmis proceselor daemon pentru repornirea acestora şi recitirea fişierului/fişierelor de configurare;
- `SIGINT` (`2`) (signal interrupt): este folosit pentru a întrerupe un proces; de obicei, efectul imediat este terminarea procesului;
- `SIGQUIT` (`3`) (signal quit): este folosit pentru închiderea unui proces; este mai puternic decât `SIGINT`: unele procese pot fi terminate cu `SIGQUIT` dar nu cu `SIGINT`;
- `SIGILL` (`4`) (signal illegal): procesul execută o instrucţiune invalidă;
- `SIGKILL` (`9`) (signal kill): cel mai puternic semnal; termina în mod necondiţionat un proces; nu poate fi ignorat;
- `SIGSEGV` (11) (signal segment violation): semnal transmis în momentul accesului invalid la o zona de memorie; rezulta, de obicei, în terminarea procesului şi la apariţia "îndrăgitului" mesaj "Segmentation fault";

- `SIGTERM` (`15`) (signal termination): termină un proces; spre deosebire de `SIGKILL`, acesta poate fi ignorat; în mod tipic nucleul transmite întâi `SIGTERM` unui proces pentru a-l anunța că va fi terminat; procesul realizează operații de curățire (cleanup), după care kernel-ul transmite `SIGKILL` şi procesul este terminat;
- `SIGSTOP` (`19`) (signal stop): suspendă procesul curent;
- `SIGCONT` (`18`) (signal continue): reia execuția unui proces suspendat.

4.4.2 Comenzile kill, killall şi pkill

Comenzile `kill`, `killall` şi `pkill` sunt utilizate pentru a transmite un semnal unui proces sau unui set de procese.

4.4.2.1 Comanda kill

Comanda `kill` transmite un semnal unui singur proces. Comanda primeşte ca argument pid-ul procesului (sau pid-urile proceselor) către care se doreşte să se transmită semnalul. În mod implicit comanda transmite semnalul `SIGTERM`:

```
razvan@asgard:~$ xcalc &
[1] 3590
razvan@asgard:~$ ps
  PID TTY          TIME CMD
 3092 pts/1    00:00:00 bash
 3590 pts/1    00:00:00 xcalc
 3591 pts/1    00:00:00 ps
razvan@asgard:~$ kill 3590
razvan@asgard:~$
[1]+  Terminated              xcalc
razvan@asgard:~$ ps
  PID TTY          TIME CMD
 3092 pts/1    00:00:00 bash
 3592 pts/1    00:00:00 ps
```

Comanda poate fi configurată pentru a transmite un anumit semnal unui proces dat. Astfel, dacă dorim transmiterea semnalului `SIGSTOP` către un anumit proces se poate utiliza una din alternativele:

```
$ kill -SIGSTOP 1234 5678
$ kill -STOP 1234 5678
$ kill -19 1234 5678
```

Am presupus că se doreşte transmiterea semnalului către procesele cu pid-urile `1234`, respectiv `5678`. Se observă că semnalul poate fi transmis ca parametru fie în formă literală (`SIGSTOP`, `STOP`), fie în formă numerică (`19`).

4.4.2.2 Comanda killall

Comanda `killall` este folosită pentru transmiterea unui semnal către un proces sau către un set de procese atunci când se cunoaşte numele procesului. Un exemplu de utilizare este prezentat mai jos:

```
asgard:/home/razvan# ps -C apache2 -H
  PID TTY          TIME CMD
 2792 ?        00:00:00 apache2
 2793 ?        00:00:00   apache2
 2794 ?        00:00:00   apache2
 2796 ?        00:00:00   apache2
asgard:/home/razvan# killall -KILL apache2
asgard:/home/razvan# ps -C apache2
  PID TTY          TIME CMD
```

În acest exemplu, s-a trimis semnalul de terminare a unui proces (`SIGKILL`) către toate procesele `apache2` din sistem. Modul de transmitere a semnalului ca argument al comenzii este acelaşi ca în cazul comenzii `kill`.

O opţiune care poate fi utilă este opţiunea `--user` care permite transmiterea unui semnal către toate procesele unui anumit utilizator:

```
asgard:/home/razvan# ps -o pid,comm,user -u guest
  PID COMMAND         USER
 3761 su              guest
 3762 bash            guest
 3802 su              guest
 3803 bash            guest
 3822 top             guest
 3825 bc              guest
asgard:/home/razvan# killall --user guest
asgard:/home/razvan# ps -o pid,comm,user -u guest
  PID COMMAND         USER
```

În situaţia de mai sus s-a transmis semnalul implicit (`SIGTERM`) către toate procesele utilizatorului `guest`, rezultând în terminarea acelor procese.

4.4.2.3 Comanda pkill

Comanda `pkill` poate fi comparată cu comanda `pgrep`. În timp ce comanda `pgrep` oferea informaţii despre anumite procese, comanda `pkill` transmite un semnal unor procese pe baza unor criterii de selecţie a acestora.

La fel ca în cazul `pgrep`, selecţia se poate realiza după pid-ul procesului părinte, după terminalul utilizat, după id-ul utilizatorului sau al grupului. Argumentul semnal se transmite la fel ca în cazul comenzilor `kill` şi `killall`.

4.4.3 Transmiterea de semnale prin combinaţii de taste

În afara comenzilor de mai sus, un utilizator poate transmite anumite semnale procesului curent (cel care rulează în acel moment în terminal) prin intermediul unor combinaţii de taste specifice.

Astfel de combinaţii sunt:

- `CTRL-Z`: transmite semnalul `SIGSTOP` către procesul curent, care are drept consecinţă suspendarea acestuia. O situaţie utilă este atunci când se rulează editorul `vi` şi se doreşte rularea altei comenzi. Se foloseşte `CTRL-Z` pentru suspendarea procesului şi revenirea la promptul terminalului, unde poate fi introdusă comanda. După introducerea comenzii se foloseşte comanda `fg` pentru a reactiva procesul `vi`.
- `CTRL-C`: transmite semnalul `SIGINT` către procesul curent, care, de obicei, va omorî procesul. De obicei, combinaţia de taste se foloseşte în cazul în care un proces este blocat.
- `CTRL-\`: transmite semnalul `SIGQUIT`. Acesta are acelaşi rol ca şi `CTRL-C`, doar că este mai puternic.

Trebuie precizat că nu toate aceste combinații de taste au efect pentru toate procesele, întrucât unele procese pot ignora semnalele transmise.

4.5 Comunicația între procese

La fel cum în cadrul unui departament mai mulți oameni conlucrează pentru a îndeplini o sarcină, în cadrul unui sistem de operare, procesele acestuia interacționează. Interacțiunea proceselor într-un sistem de operare poartă numele de comunicație între procese.

De ce este nevoie de comunicație între procese? În primul rând un proces are nevoie de resursele pe care i le pune la dispoziție un alt proces. În cazul utilitarului `top`, acesta va culege informații pe care i le furnizează alte procese prin intermediul nucleului sistemului și al sistemului de fișiere `procfs`.

În altă situație, un proces trebuie să aștepte ca un alt proces să își termine o sarcină pentru ca el să poată continua. Aceasta formă de comunicație se numește sincronizare: un proces trebuie să aștepte un alt proces.

Comunicația între procese este intermediată în diverse moduri: cu ajutorul unui fișier, a unui socket, a unei zone de memorie sau a unui canal de comunicație (pipe). Intermedierea printr-un fișier este destul de simplă: un proces scrie o informație într-un fișier, iar un alt proces o citește. Nu vom prezenta comunicația prin intermediul unui socket sau a unei regiuni de memorie, întrucât depășesc aria de cuprindere a acestui capitol. Vom discuta, însă, despre comunicația prin intermediul unui canal de comunicație (pipe).

4.5.1 Operatorul | (pipe)

Operatorul | (pipe) este folosit pentru a asigura comunicația între două procese rulate din linia de comandă folosind un canal de comunicație (pipe).

Comunicația prin intermediul unui pipe este o îmbunătățire a comunicației prin intermediul unui fișier. Astfel, în cazul comunicației prin intermediul unui fișier, un proces scrie ieșirea într-un fișier iar un alt proces folosește acel fișier ca intrare. În cazul comunicației prin pipe, ieșirea primului proces este folosită ca intrare pentru al doilea fără a mai fi nevoie de un fișier.

Vom folosi pentru exemplificare comanda `grep` care caută un cuvânt în cadrul unui fișier. Dacă, spre exemplu, dorim aflarea de informații despre procesele `bash` din sistem folosind `grep`, ar trebui, în primă fază, să redirectăm ieșirea comenzii `ps` într-un fișier, iar apoi să folosim `grep` pe acel fișier, ca în exemplul de mai jos:

```
razvan@asgard:~$ ps -e > ps.out
razvan@asgard:~$ grep bash ps.out
 3029 pts/0    00:00:00 bash
 3092 pts/1    00:00:00 bash
 3299 pts/2    00:00:00 bash
 3317 pts/2    00:00:00 bash
 3740 pts/3    00:00:00 bash
 3758 pts/3    00:00:00 bash
 3781 pts/4    00:00:00 bash
 3799 pts/4    00:00:00 bash
```

Același rezultat ca mai sus poate fi însă realizat prin intermediul operatorului `|`, fără a mai fi nevoie de intermedierea printr-un fișier:

```
razvan@asgard:~$ ps -e | grep bash
 3029 pts/0    00:00:00 bash
 3092 pts/1    00:00:00 bash
 3299 pts/2    00:00:00 bash
 3317 pts/2    00:00:00 bash
 3740 pts/3    00:00:00 bash
 3758 pts/3    00:00:00 bash
 3781 pts/4    00:00:00 bash
 3799 pts/4    00:00:00 bash
```

Pentru a exemplifica puterea acestui operator vom considera următoarea problemă: dorim să aflăm primii 3 utilizatori din sistem care au directorul de bază în `/home`, ordonați în ordinea alfabetică a numelui de utilizator.

Informații despre utilizatorii din sistem se găsesc în `/etc/passwd`. Vom folosi grep pentru a extrage utilizatorii care au directorul de bază în `/home`:

```
razvan@asgard:~$ cat /etc/passwd | grep /home
razvan:x:1000:1000:razvan,,,:/home/razvan:/bin/bash
haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/false
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
ftp:x:107:65534::/home/ftp:/bin/false
```

În continuare vom sorta rezultatul obținut, după care vom reține numai primele 3 intrări:

```
razvan@asgard:~$ cat /etc/passwd | grep /home | sort
ftp:x:107:65534::/home/ftp:/bin/false
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/false
razvan:x:1000:1000:razvan,,,:/home/razvan:/bin/bash
razvan@asgard:~$ cat /etc/passwd | grep /home | sort | head -3
ftp:x:107:65534::/home/ftp:/bin/false
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/false
```

4.6 Swapping

Un concept important în studiul sistemelor de operare şi al proceselor acestora este conceptul de **swapping**. Swapping este o măsură de a compensa cantitatea insuficientă de memorie RAM de care dispune un sistem la un moment dat. Pentru aceasta, un sistem de operare va folosi o porțiune din disc.

În cazul unui sistem încărcat (cu multe procese), memoria RAM se poate dovedi insuficientă pentru a satisface toate procesele. În această situație o parte din paginile de memorie sunt evacuate (swappat-e) pe disc pentru a face loc pentru paginile de memorie utile în momentul de față. În momentul în care paginile evacuate sunt din nou necesare, se vor evacua alte pagini pentru a face loc celor curente.

Principiul swapping-ului este prezentat în figura de mai jos:

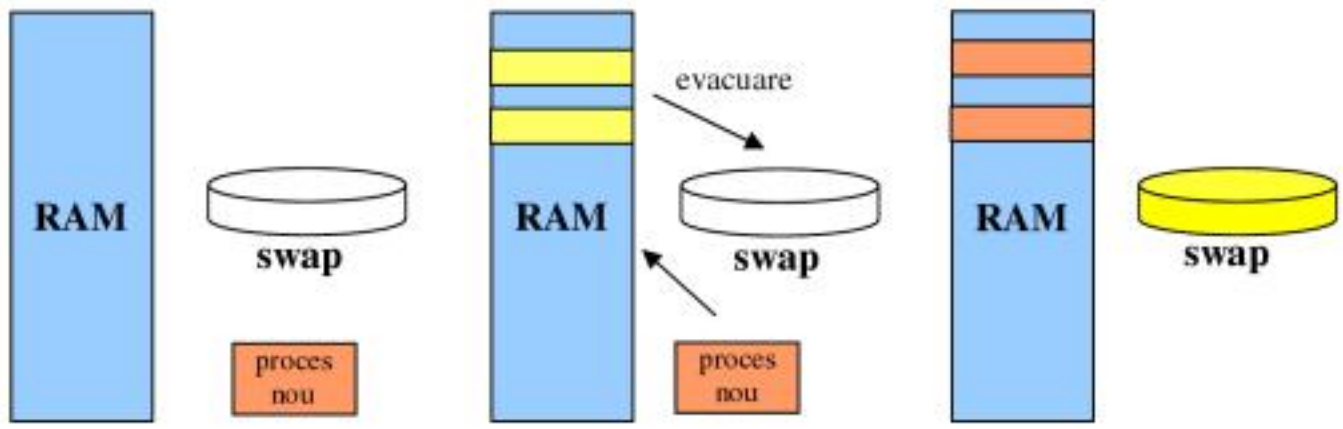


Figura 4-3 Swapping

Pe un sistem Linux, există o partiție dedicată (partiția de swap – [vezi 2.2.2]), care este folosită în cazul în care memoria din sistem este insuficientă. În cazul Windows, spațiul de swap este o parte din partiția de sistem. Pentru a afla informații despre spațiul de swap utilizat se poate folosi comanda `free` (aceste informații se găsesc şi în zona de sumarizare din ecranul `top`):

```
razvan@asgard:~$ free
             total       used       free     shared    buffers     cached
Mem:        514748     336876     177872          0      19892     185200
-/+ buffers/cache:     131784     382964
Swap:       497972          0     497972
```

Se observă că sistemul beneficiază de suficient de multă memorie RAM (`512 MB`) şi nu este folosit spațiul de swap.

4.7 Studii de caz

4.7.1 Managementul proceselor/serviciilor pe Windows

În Windows, un proces are caracteristici asemănătoare cu procesele din Linux. Totuşi, deşi un proces este creat prin intermediul altui proces, nu există o ierarhie de procese şi nici noțiunea directă de proces părinte şi proces fiu. La fel ca în Linux, un proces este identificat printr-un PID, are asociate zone de memorie şi fişiere deschise, etc.

4.7.1.1 Task manager

Interfața de vizualizare şi gestiune a proceselor în Windows este asigurată de `Task Manager`. Pentru pornirea acestuia se foloseşte combinația de taste `CTRL-ALT-DEL sau CTRL-SHIFT-ESC` sau click dreapta pe bara centrală şi selectarea opțiunii `Task Manager` din meniu.

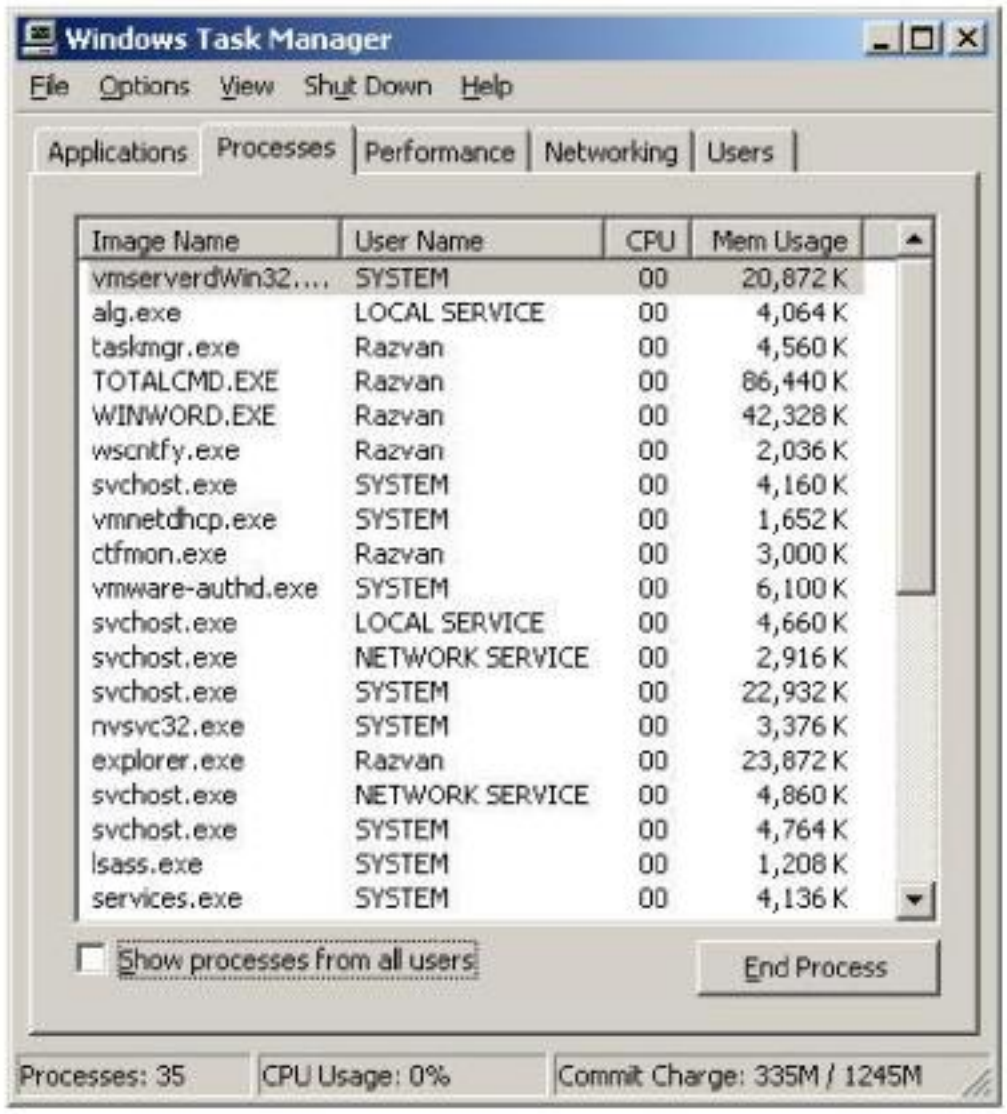


Figura 4-4 Windows Task Manager

`Task Manager` poate fi comparat cu utilitarul `top`. Afişează informaţii despre procese şi despre sistem şi poate fi configurat pentru personalizarea afişării sau pentru interacţiunea cu procesele din sistem.

În mod implicit, `Task Manager` afişează numele programului care a generat procesul, utilizatorul acestuia, procentajul de procesor folosit şi memoria folosită. Dacă se doreşte afişarea altor câmpuri se accesează meniul `View->Select Columns`. Se pot afişa PID-ul procesului, memoria virtuală utilizată, maximul de memorie utilizat, numărul de fişiere deschise, numărul de thread-uri, etc.

În afara vizualizării proceselor din sistem, `Task Manager` mai oferă următoarele servicii:

- permite crearea unui nou proces prin introducerea imaginii de executabil asociate (`File->New Task (Run...)`);
- permite monitorizarea şi analiza performanţei sistemului, prin accesarea tab-ului `Performance`; informaţiile de aici sunt asemănătoare cu cele oferite de top în zona de sumarizare;
- permite deconectarea unui utilizator, sau oprirea, repornirea sau suspendarea sistemului (meniul `Shut Down`);
- permite terminarea unui proces; se selectează procesul dorit şi se apasă butonul `End Process`.

Se observă că `Windows Task Manager` oferă cea mai mare parte din facilităţile furnizate de comenzile Unix de interacţiune cu procese sistemului.

Un utilitar puternic de vizualizare şi control a proceselor pe Windows este `ProcessExplorer` (http://www.microsoft.com/technet/sysinternals/utilities/ProcessExplorer.mspx)

4.7.1.2 Services

În Windows, procesele daemon poartă numele de **servicii**. În Windows, ca şi în Linux, serviciile sunt procese care rulează în background şi sunt folosite pentru monitorizarea şi întreţinerea sistemului.

Fereastra de interacţiune cu serviciile Windows este afişată, în Windows XP, prin accesarea `Start->Control Panel->Administrative Tools->Services` sau click dreapta pe `My Computer->Manage->Services`.

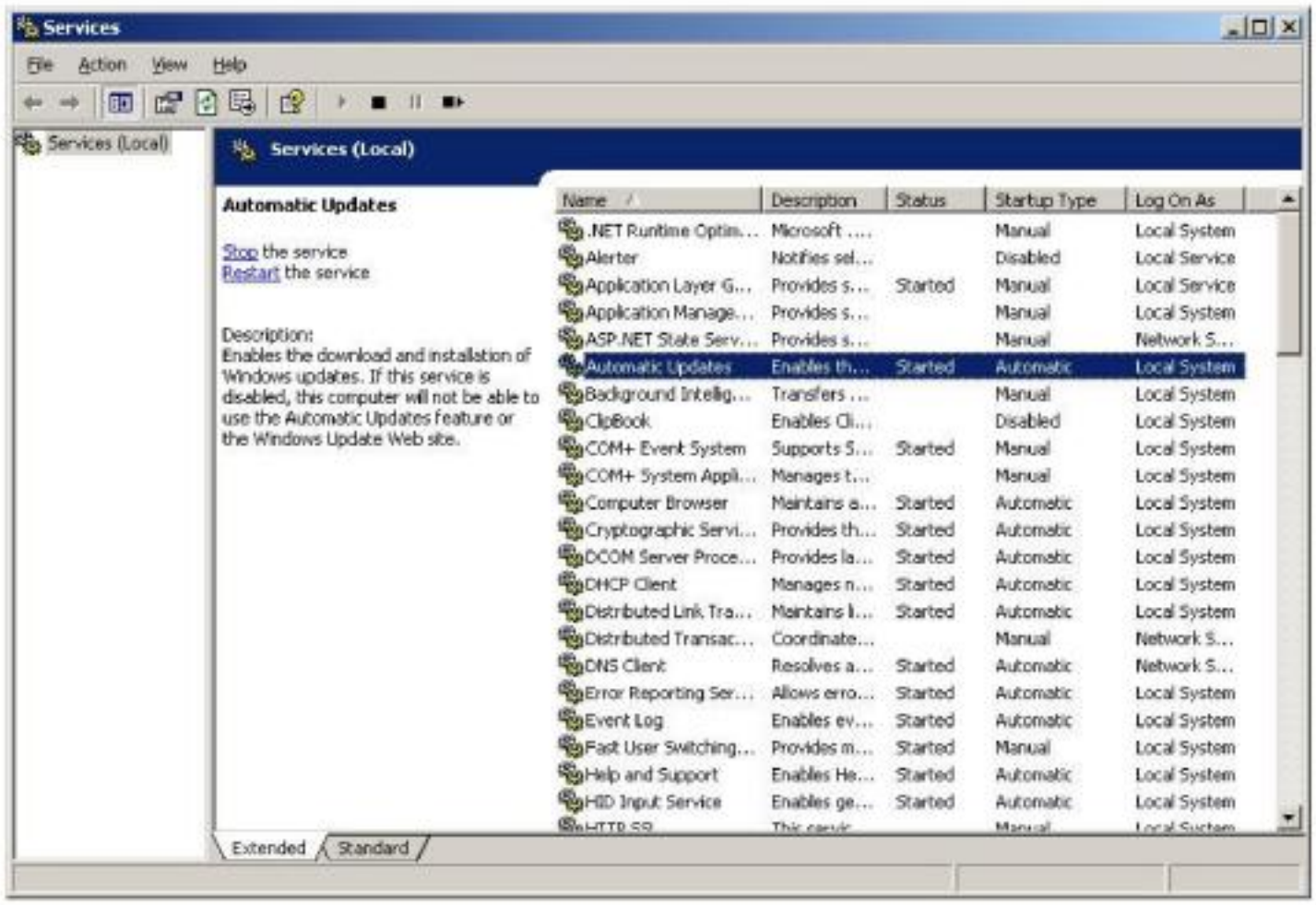


Figura 4-5 Windows Services

Ecranul afişat prezintă toate serviciile sistemului. Coloanale afişate indică numele serviciului, o scurtă descriere, starea lui (oprit sau pornit), modul de pornire a serviciului (dezactivat, manual sau automat şi, la Windows Vista, întarziat). Pentru pornirea, oprirea sau repornirea serviciul ca şi pentru schimbarea modului de pornire se accesează meniul contextual al serviciului (click dreapta) şi se selectează opţiunea `Properties`:

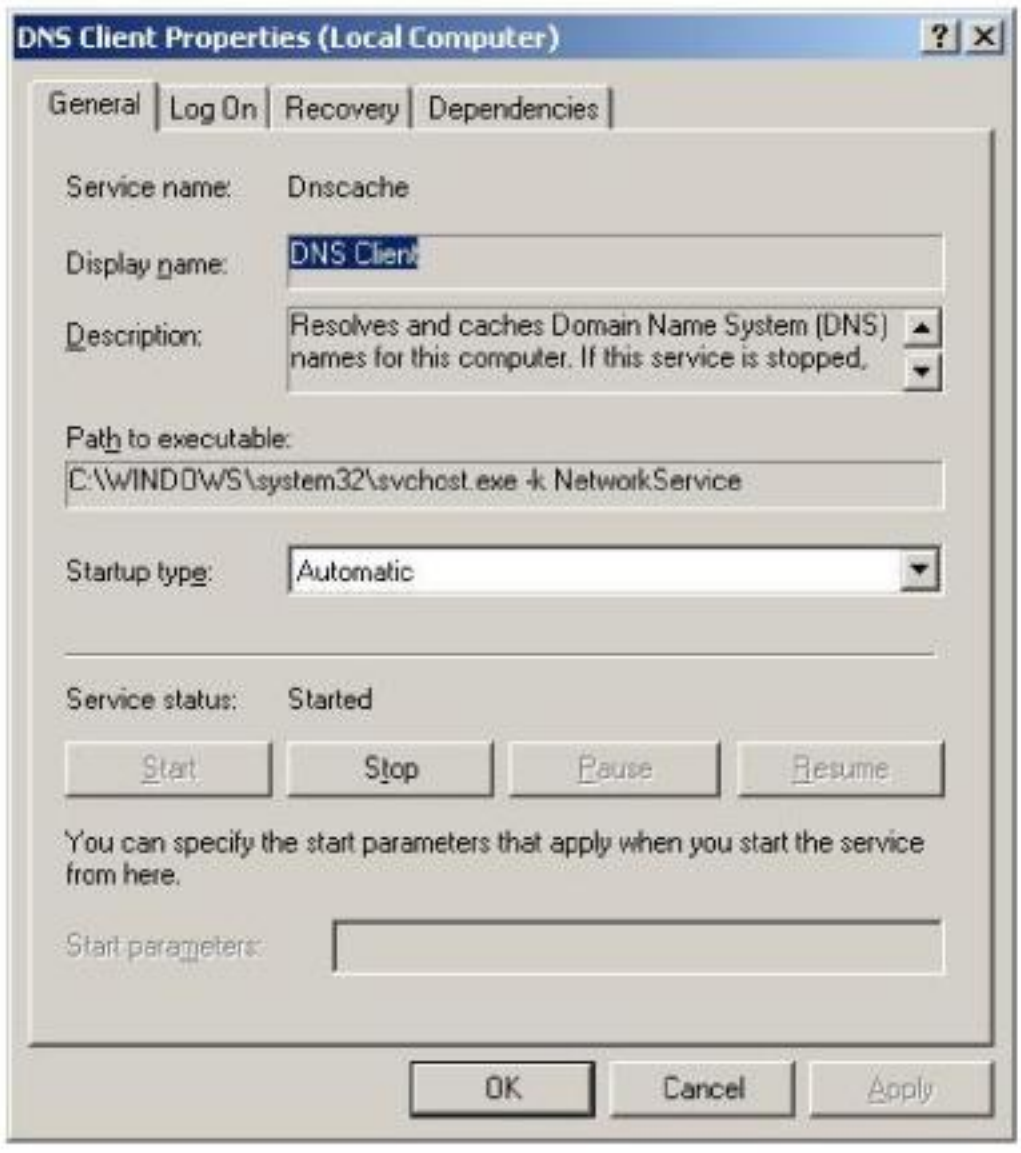


Figura 4-6 Interfața de control a unui serviciu Windows

4.7.2 Procese importante

Orice sistem de operare deține un set de procese cu rol important în rularea și întreținerea sistemului de operare.

4.7.2.1 Procese importante Linux

Procese importante în Linux sunt:

- `init`: este procesul cu pid-ul 1 în sistemele Unix și este părintele tuturor proceselor; rolul său nu se limitează doar la a porni celelalte procese importante în sistem ci și de a deveni procesul părinte pentru procesele rămase orfane;
- `kswapd` (kernel swap daemon): este responsabil cu alegerea paginilor de memorie care vor fi evacuate pe disc în momentul în care memoria este insuficientă;
- `pdflush` (page daemon): este responsabil cu asigurarea consistenței buffer-elor folosite de sitemul de fișiere;
- `getty`: este procesul resposabil cu autentificarea în sistem în terminalele virtuale; `getty` oferă promptul `login :` utilizatorului;
- `Xorg`: este procesul resposabil cu pornirea interfeței grafice; are asociat terminalul virtual `tty7` (și poate fi accesat prin combinația `ALT+CTRL-F7`).

4.7.2.2 Procese importante Windows

În Windows procese importante sunt:

- lsass (Local Security and Authentication Server) verifică validitatea autentificării utilizatorilor; în cazul unei autentificări corecte generează un jeton folosit pentru deschiderea unei prime sesiuni shell pe sistem; lsass este şi numele unui virus;
- svchost (Generic Host Process for Win32 Services) este procesul folosit pentru gestiunea serviciilor din Windows;
- csrss (client/server run-time subsystem) este responsabil pentru crearea ferestrelor de consolă şi gestiunea thread-urilor;
- winlogon este o componentă esențială a subsistemului de autentificare în Windows;
- explorer este este shell-ul unui sistem Windows, însă cu interfață grafică.

Mai multe informații despre procesele importante Windows găsiți la http://www.neuber.com/taskmanager/process/

4.7.3 Prioritatea unui proces

În sistemele de operare moderne, procesele au asociată o prioritate. Aceasta influențează probababilitatea ca un proces să fie planificat pe procesor. De obicei, procesele care se blochează des (denumite I/O intensive) au asociate o prioritate mai mare, în timp ce procesele care utilizează intens procesorul (CPU intensive), au o prioritate mai mică.

Prioritatea proceselor nu este fixă. Sistemul de operare poate modifica prioritatea proceselor pentru a asigura echitatea planificării. Totodată, utilizatorul poate modifica prioritatea unui proces dacă doreşte acest lucru.

4.7.3.1 Schimbarea priorității unui proces în Linux

În Linux, prioritatea unui proces poartă numele de **niceness** (cât de echitabil se comportă un proces raportat la celelalte). Un proces are o prioritate mai mare cu cât valoarea sa nice este mai mică. Valorea nice pentru un proces se încadrează în limitele `[-20, 19]`. `-20` înseamnă un proces maxim prioritar, în timp ce `19` înseamnă un proces minim prioritar.

Prioritatea unui proces într-un sistem Linux se află folosind `ps` şi opțiunea `nice`:

```
razvan@asgard:~$ ps -e -o pid,tty,comm,nice --sort nice
  PID TT       COMMAND          NI
    3 ?        events/0         -5
    4 ?        khelper          -5
    5 ?        kthread          -5
    8 ?        kblockd/0        -5
    9 ?        kacpid           -5
  [...]
    1 ?        init              0
  [...]
 3799 pts/4    bash              0
 4009 pts/1    ps                0
    2 ?        ksoftirqd/0      19
```

Majoritatea proceselor pornesc cu prioritatea 0. Pentru a modifica prioritatea de start a unui proces se foloseşte comanda `nice`, folosind argumentul `-n`:

```
razvan@asgard:~$ nice -n -10 ps -e -o pid,tty,comm,ni
nice: cannot set niceness: Permission denied
razvan@asgard:~$ nice -n 15 ps -C ps -o pid,tty,comm,ni
  PID TT       COMMAND          NI
 4051 pts/1    ps               15
asgard:/home/razvan# nice -n -10 ps -C ps -o pid,tty,comm,ni
  PID TT       COMMAND          NI
 4048 pts/2    ps              -10
```

Se observă că un utilizator neprivilegiat nu poate stabili o prioritate negativă (puternică), dar poate stabili una pozitivă (mai slabă).

Schimbarea în timp real a priorității unui proces se realizează prin intermediul `top`. În ecranul `top` se folosește tasta `r` (renice). Zona de comandă oferă promptul `PID to renice:` în care se introduce pid-ul procesului a cărui prioritate va fi schimbată. După aceasta se va cere introducerea unei valori nice asociată procesului. Un utilizator neprivilegiat poate doar incrementa valoarea nice a unui proces (îl va face mai puțin prioritar).

4.7.3.2 Schimbarea priorității unui proces în Windows

În Windows, utilizatorul poate stabili prioritatea unui proces la niște valori cu granularitate mai mare: `High`, `Very High`, `Normal`, etc. Pentru aceasta se folosește tot interfața pusă la dispoziție de `Task Manager`. Pentru alterarea priorității unui proces, se realizează click dreapta pe intrarea asociată procesului, se alege opțiunea `Set priority` și apoi se stabilește prioritatea dorită.

Exemplificarea este realizată în figura de mai jos:

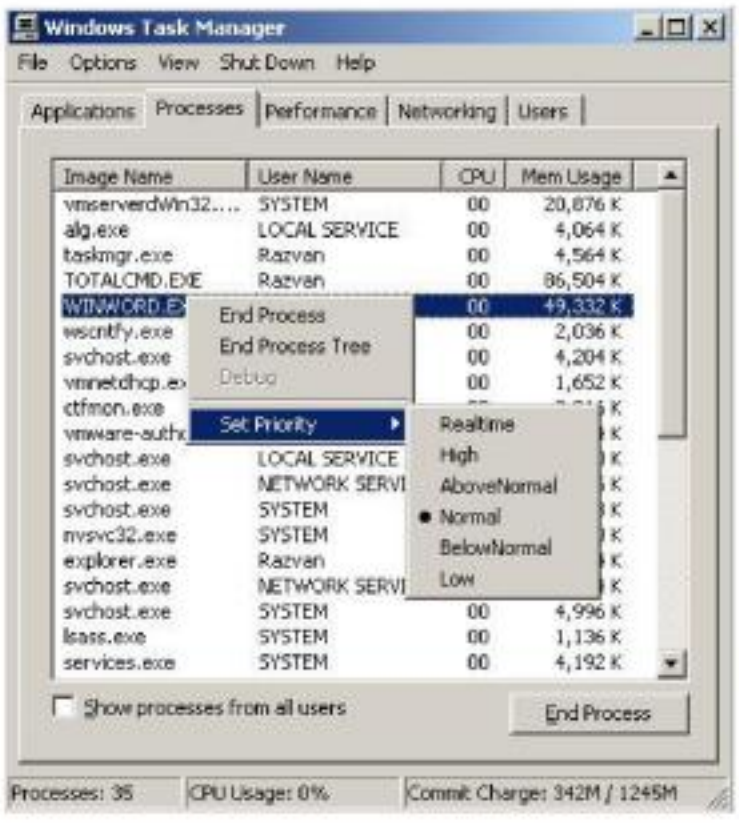


Figura 4-7 Stabilirea priorității unui proces în Windows

Cuvinte cheie

- proces
- program
- pid
- procfs
- interactivitatea unui proces
- context switch
- cuantă de timp
- multitasking
- ierarhie de procese
- init
- ps
- pstree
- pgrep
- top
- background
- foreground
- operatorul &
- bg, fg
- suspendarea unui proces
- terminal
- proces daemon
- nohup
- semnal
- kill
- killall, pkill
- CTRL-Z, CTRL-C, CTRL-\, CTRL-X
- operatorul |
- Task manager
- Services
- init, kswapd, pdflush, getty
- lsass, svchost, csrss
- prioritatea unui proces
- nice

Întrebări

Care din următoarele este un proces important Unix?

- ps
- SIGQUIT
- init
- lsass

Care utilitar nu poate fi folosit pentru a transmite semnale către un proces?

- kill
- top
- nohup
- killall

Câte procese init pot exista la un moment dat într-un sistem Linux?
- 1
- câte unul per procesor
- niciunul
- oricâte

Câte procese bash pot exista la un moment dat într-un sistem Linux?
- 1
- câte unul per procesor
- niciunul
- oricâte

Care din următoarele comenzi nu afişează PID-ul unui proces?
- ps
- pgrep
- kill
- top

În Linux orice program poate fi imaginea unui singur proces. Comanda top nu afişează pid-urilor proceselor din sistem.
- adevărat, adevărat
- adevărat, fals
- fals, adevărat
- fals, fals

Care comandă poate produce aceleaşi efecte ca apăsarea combinației de taste CTRL-Z?
- pkill
- ps
- operatorul &
- fg

Ce locație din sistemul de fişiere oferă informații despre procese?
- /sys
- /proc
- /home
- /

În Linux procesele nu au pid-uri negative. În Linux procesele nu au priorități negative.
- adevărat, adevărat
- adevărat, fals
- fals, adevărat
- fals, fals

Care asociere nu este validă?
- Task Manager – top
- serviciu – daemon
- explorer – nice
- lsass – getty

5 Pornirea şi iniţializarea sistemului

The term reboot comes from the middle age (before computers). Horses who stopped in mid-stride required a boot to the rear to start again. Thus the term to rear-boot, later abbreviated into reboot

Ce se învaţă din acest capitol?

- Pornirea sistemului; conceptul de bootstrapping
- Noţiunea de multitasking
- Fazele pornirii sistemului
- BIOS (POST, CMOS)
- Dispozitive boot-abile, sector bootabil (MBR/VBR)
- Rolul unui bootloader
- GRUB: rulare şi configurare
- Încărcarea nucleului Linux
- Iniţializarea unui sistem Linux: `init`/`upstart` şi pornirea serviciilor
- Pornirea şi iniţializarea Windows
- Noţiuni de interoperabilitate Linux-Windows

5.1 Pornirea sistemului

Pornirea şi iniţializarea sistemului se referă la mecanismele prin care sistemul de operare şi aplicaţiile de bază ajung să fie încărcate şi folosite de utilizator.

Pornirea sistemului se realizează, de obicei, prin apăsarea unui buton specific de pe calculator. În sens mai larg, ne vom referi la pornirea sistemului ca acţiunea de iniţializare hardware şi sofware. Astfel, din punct de vedere fizic, pornirea sistemului cuprinde acţiunea de pornire (*power on*) cât şi repornire (reboot). **Pornirea** (*power on*) se referă la activarea sistemului în momentul în care acesta era oprit (prin intermediul butonului de pornire – *power on*). **Repornirea** sistemului se referă la reactivarea sistemului. Acesta este pornit, eventual rulează o instanţă a unui sistem de operare, după care este repornit (fie prin intermediul butonului Reset, fie printr-o comandă din sistemul de operare care deja rulează: de exemplu comanda `reboot` în Linux).

Pornirea sistemului presupune, aşadar, iniţializarea hardware şi software a sistemului. Iniţializarea hardware presupune verificarea şi configurarea componentelor hardware, pe când iniţializarea software presupune încărcarea sistemului de operare. Nu vom considera rularea anumitor aplicaţii peste sistemul de operare ca făcând parte din iniţializarea software.

Pornirea şi iniţializarea sistemului prezintă un set de faze pe care le vom preciza în continuare.

5.1.1 Problematica pornirii sistemului – bootstrapping

Pornirea sistemului şi încărcarea sistemului de operare poartă numele de ***booting*** *sau de* ***bootstrapping****.*

Denumirea de bootstrapping îşi are originea în *Baronul de Münchhausen*. Într-o poveste a acestuia, el reuşeşte să se salveze de la înec trăgându-se de baierele cizmelor (*boot* = cizmă, *strap* = baieră). Analogia cu această poveste se reflectă în faptul că sistemul de operare trebuie încărcat fără existenţa unui alt sistem de operare.

Altfel spus, se pune întrebarea: *Cum putem încărca în memorie un program dacă nu avem un alt program cu ajutorul căruia să-l încărcam?* Aparent, trebuie „să ne tragem de baierele cizmelor" pentru „a ne scoate din apă". Vom vedea, însă, că există un set de faze care se vor parcurge succesiv pentru încărcarea a diverselor componente până la încărcarea completă a sistemului de operare.

5.1.2 Fazele pornirii sistemului

Pornirea sistemului presupune încărcarea în memorie a diverselor componente până la încărcarea completă a sistemului de operare. Fazele importante ale pornirii sistemului sunt:

- **încărcarea BIOS-ului**: presupune încărcarea unui program utilizat pentru iniţializarea componentelor hardware;
- **încărcarea bootloader-ului**: presupune încărcarea unui program destinat optării între două sisteme de operare care coexistă pe acelaşi sistem fizic;
- **încărcarea nucleului**: presupune încărcarea nucleului (kernel-ului) sistemului de operare;
- **iniţializarea sistemului de operare**: presupune pornirea principalelor procese care asigura utilizarea sistemului de operare.

Fazele prezentate sunt ordonate „cronologic": o fază este rulată şi este responsabilă pentru rularea fazei ulterioare. Vom descrie, în cadrul acestui capitol, fiecare din fazele prezentate şi interacţiunea între acestea.

5.1.3 BIOS

BIOS-ul sau *Basic Input Output System* este primul program încărcat în momentul pornirii sistemului. BIOS-ul este responsabil cu verificarea şi iniţializarea componentelor hardware.

BIOS-ul este un exemplu de **firmware**. Firmware-ul este o componentă software distribuită sub formă binară pe un suport de memorie nevolatilă. În cazul nostru, BIOS-ul rezidă pe un chip de memorie ROM de pe placa de bază. În momentul în care sistemul este pornit sau repornit, se rulează automat din chip-ul ROM de pe placa de bază BIOS-ul. BIOS-ul cunoaşte specificul hardware al sistemului şi devine responsabil cu iniţializarea componentelor fizice ale sistemului.

După verificarea şi iniţializarea componentelor, BIOS-ul este responsabil cu încărcarea bootloader-ului.

5.1.3.1 POST

Etapa de verificare şi de iniţializarea componentelor hardware ale sistemului (procesor, memorie, placă video) poartă numele de POST (*Power-on Self Test*). Dacă una dintre componente prezintă erori şi nu poate fi iniţializată corespunzător, atunci BIOS-ul va emite un sunet specific şi nu se continuă iniţializarea sistemului.

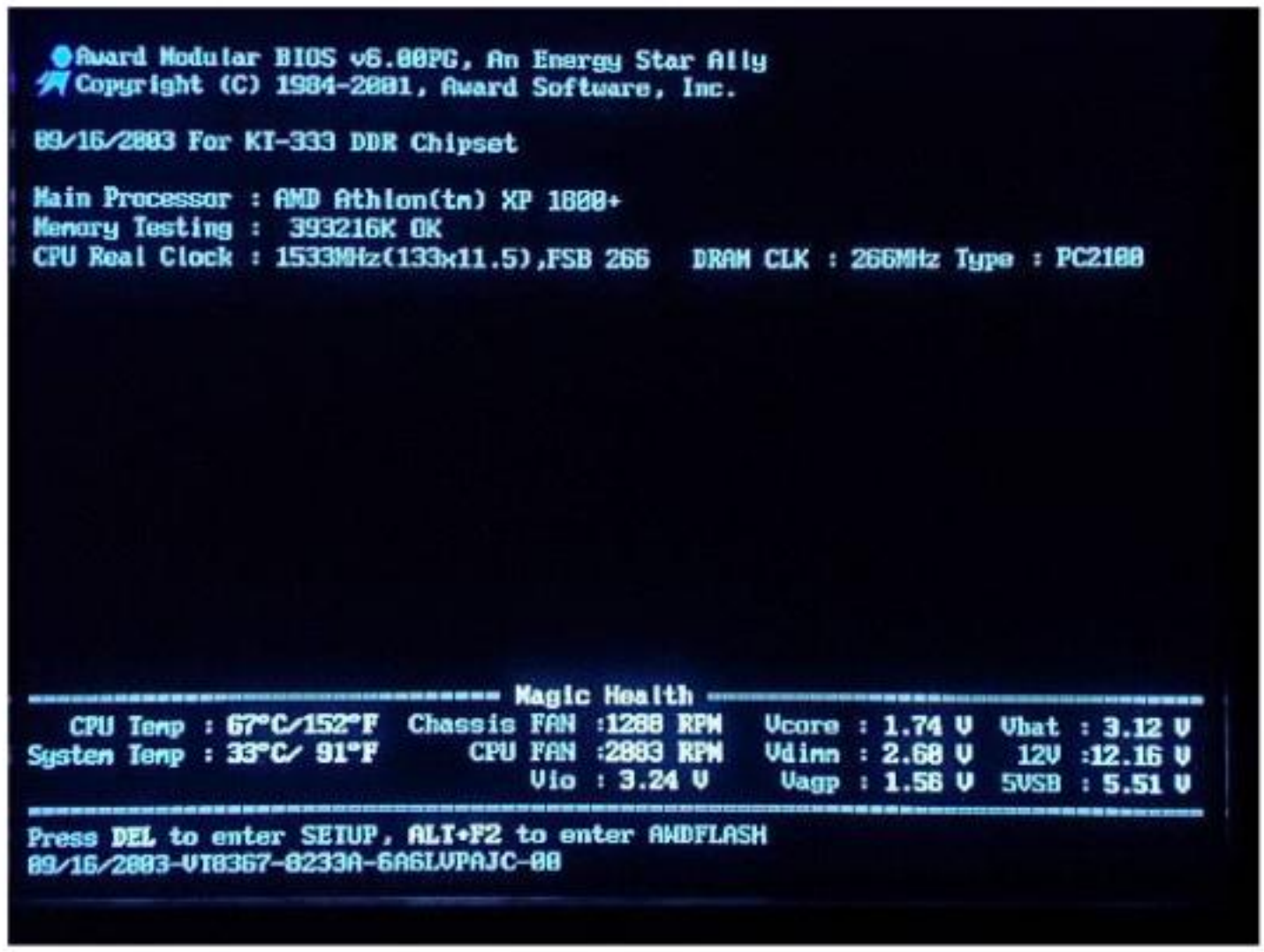


Figura 5-1 BIOS POST (Power-On Self Test)

5.1.4 CMOS

Deşi un program mic şi limitat la specificul hardware al sistemului, BIOS-ul poate fi configurat. Configurările BIOS-ului sunt păstrate într-un chip separat de memorie RAM nevolatilă alimentată prin intermediul unei baterii. Atât chip-ul cât şi bateria se găsesc pe placa de bază. În momentul în care bateria este scoasă, configurările BIOS-ului se pierd revenindu-se la configurările implicite.

Figura 5-2 Ecran de configurare CMOS

Configurarea BIOS-ului se realizează prin intermediul unui ecran de configurare denumit **CMOS**. Configurațiile posibile de aici includ stabilirea unei parole pe BIOS, selecția dispozitivului de boot [vezi 5.2.1], alterarea frecvenței procesorului și alte configurări care depind de placa de bază și de componentele hardware existente în sistem.

5.2 Bootloader

Bootloader-ul este componenta utilizată pentru selecția între mai multe sisteme de operare și pentru transmiterea unor opțiuni de încărcare a nucleului sistemului de operare. Bootloader-ul este încărcat în memorie de BIOS.

Bootloader-ul se regăsește pe primul sector al unui dispozitiv boot-abil, denumit și **sectorul de boot**. Aceasta se întâmplă întrucât BIOS-ul poate încărca în memorie un singur sector (`512` octeți). Se poate întâmpla ca bootloader-ul să fie suficient de mare încât să nu poată încăpea în primul sector al dispozitivului boot-abil, astfel că se va încărca o a doua etapă (*second stage*) a bootloader-ului, rezidentă pe un spațiu mai mare de pe hard-disk. Un astfel de **multiple-stage bootloader** este GRUB, bootloader-ul implicit pe majoritatea distribuțiilor Linux.

5.2.1 Dispozitive boot-abile

Un dispozitiv boot-abil este un dispozitiv al cărui prim sector este un **sector boot-abil**. Sectorul boot-abil se caracterizează prin faptul că ultimii doi octeți sunt `0xAA55` (din cei `512`).

Exemple de dispozitive boot-abile sunt CD-ROM-uri, hard-disk-uri, floppy disk-uri, USB flash, etc. În cazul acestor dispozitive, dacă primul sector are structura specifică unui sector boot-abil atunci poate fi folosit pentru încărcarea altor informații existente.

Figura 5-3 Alegerea dispozitivului de boot

După rularea POST, BIOS-ul consultă CMOS şi urmează ordinea de boot-are de acolo. Astfel, dacă CMOS-ul a fost configurat ca în imaginea de mai sus, BIOS-ul va efectua următoarele operaţii:

- va verifica existenţa unui CD în CD-ROM drive; dacă acesta există va verifica existenţa sectorului de boot;
- dacă CD-ul are un prim sector bootabil, atunci se va încărca şi se va executa conţinutul sectorului de boot; exemple de CD boot-abil este CD-ul de instalare a unui sistem de operare sau un LiveCD;
- dacă CD-ul nu are primul sector boot-abil se trece la investigaţia hard-disk drive-ului din sistem;
- dacă HDD-ul are un prim sector bootabil, se va încărca şi se va executa conţinutul sectorului; aici va rezida, de obicei, bootloader-ul; în urma execuţiei se va încărca sistemul de operare;
- dacă nici HDD-ul nu are un prim sector bootabil atunci se trece la următorul dispozitiv boot-abil (care este dezactivat - *disable*);
- deoarece nu există nici un CD boot-abil, nu se poate încărca nici un sistem de operare; BIOS-ul va afişa un mesaj specific fără a executa nimic.

5.2.1.1 Structura sectorului de boot pentru un dispozitiv bootabil

După cum am specificat anterior, un dispozitiv este boot-abil dacă sectorul de boot al acestuia este un sector boot-abil. Un sector are `512` octeţi. Un sector boot-abil are ultimii doi octeţi `0xAA55`, pentru a semnifica faptul că sectorul conţine secvenţe de cod executabile. Primul sector al unui dispozitiv bootabil poartă denumirea de **MBR** (*Master Boot Record*). Primul sector al unei partiţii al unui dispozitiv boot-abil partiţionat poartă numele de **VBR** (*Volume Boot Record*).

Sectorul de boot poate conţine instrucţiuni pentru încărcarea unei secvenţe mai mari de cod (cum este sectorul de boot al unui CD), sau poate conţine un bootloader pentru încărcarea sistemului de operare (cum este sectorul de boot al unui hard-disk).

Conținutul sectorului de boot este încărcat în memorie de BIOS. După aceasta, BIOS-ul transmite controlul programului rezident în sectorul de boot.

În figura de mai jos este prezentată structura MBR:

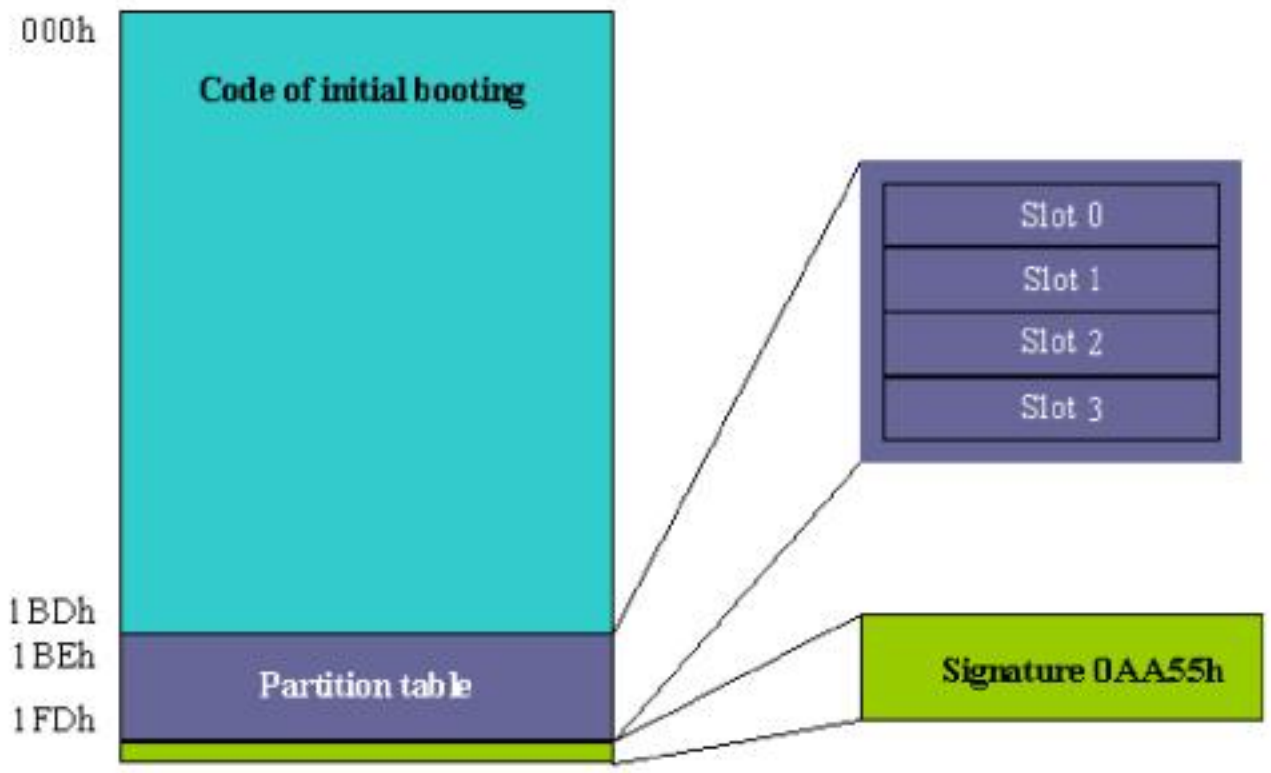


Figura 5-4 Structura MBR (Master Boot Record)

După cum se observă, avem următoarele zone importante:

- zona de cod executabil; aceasta conține programul căruia îi este transmis controlul de BIOS;
- tabela de partiții conține informații despre partițiile din sistem; fiecărei partiții îi corespunde o intrare de `16` octeți;
- codul de sector boot-abil (`0xAA55`).

Zona de cod executabil conține programul care va fi rulat după încărcarea MBR-ului în memorie de BIOS. De obicei, acest program este responsabil cu descoperirea partiției active şi încărcarea primului sector (VBR) de pe aceasta. Totuşi, în cazul existenței mai multor sisteme de operare, zona de cod poate conține codul de bootloader (sau o parte a acestuia) care poate permite utilizatorului să opteze pentru încărcarea unui sistem de operare aflat pe altă partiție decât cea de boot.

Se cuvine să prezentăm câteva detalii despre **tabela de partiții**.

Tabela de partiții

Tabela de partiții rezidă în MBR şi ocupă `64` de octeți. Un hard-disk poate conține doar patru partiții (aşa numitele partiții primare), rezultând `16` octeți pentru descrierea fiecărei partiții.

Cei `16` octeți ai fiecărei partiții oferă, pe lângă adresa de start (exprimată în *cylinder-head-sector* - http://en.wikipedia.org/wiki/Cylinder-head-sector) şi dimensiune, şi tipul partiției. O partiție poate fi activă (boot-abilă) sau non-boot-abilă. O partiție activă este aleasă de programul aflat în zona de cod a MBR pentru a continua încărcarea sistemului. Acel program va încărca primul sector (VBR) al acelei partiții.

O altă clasificare a partițiilor este clasificarea în partiții primare şi partiții extinse. Numărul de 4 partiții care pot fi descrise în tabela de partiții din MBR este, de multe ori, insuficient. Pentru a rezolva acest neajuns, a apărut noțiunea de **partiție extinsă**. Pe un sistem poate exista

o singură partiţie extinsă astfel că putem avea fie maxim 4 partiţii primare, fie o partiţie extinsă şi maxim 3 primare.

Partiţiile extinse pot conţine mai multe **partiţii logice**, mărind astfel numărul de partiţii posibile din sistem. Descrierea partiţiei extinse se regăseşte în primul sector din acea partiţie denumit **EPBR** (*Extended Partition Boot Record*).

Partiţiile sunt folosite pentru separaţia informaţiilor într-un sistem de operare. Pentru a asigura existenţa unui sistem de fişiere pe o partiţie, acea partiţie va trebui **formatată**. [vezi 2.3.3]

Salvarea sectorului de boot (backup)

Uneori se doreşte salvarea sectorului de boot al unui disc sau al unei partiţii pentru situaţia în care acesta este corupt şi trebuie reparat. În Linux, acest lucru se poate realiza foarte uşor prin intermediul comenzii `dd` şi a dispozitivelor de tip bloc [vezi 6.6]

Astfel, dacă vrem să realizăm o copie locală a MBR vom folosi utilitarul `dd` şi dispozitivul asociat hard-disk-ului (în cazul nostru `/dev/hda`). Vom copia `512` octeţi adică un sector (contorul de sectoare va fi `1`):

```
asgard:~# dd if=/dev/hda of=mbr.out count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.035273 seconds, 14.5 kB/s
asgard:~# ls -l mbr.out
-rw-r--r-- 1 root root 512 2007-08-07 23:28 mbr.out
```

5.2.2 Mecanismul de funcţionare a unui bootloader

După cum s-a specificat, un bootloader (sau bootstrap loader) este un program folosit pentru a încărca sistemul de operare, sau alte componente care vor încărca (la rândul lor) sistemul de operare.

Codul bootloader-ului este supus unor constrângeri destul de rigide, întrucât trebuie să încapă în MBR. Mai exact, este limitat la `446` de octeţi (restul fiind ocupaţi de tabela de partiţii şi de semnătura de boot `0xAA55`).

Drept urmare, bootloader-ele lucrează tipic în două faze (*stages*). Prima fază este cea care rezidă în MBR. Aceasta are rolul de a încărca de pe hard-disk a doua fază care este suficient de complexă pentru a putea începe încărcarea efectivă a sistemului de operare. În cea de-a doua fază se oferă utilizatorului un ecran de selecţie pentru situaţia în care coexistă două sisteme de operare pe acelaşi sistem fizic. Unele bootloader-e (spre exemplu GRUB) au o fază intermediară – `1.5` – pentru a permitea citirea de informaţii de pe hard-disk dincolo de o anumită limită.

După acest pas, bootloader-ul va încărca nucleului sistemului de operare (sau o parte a acestuia) şi va transmite controlul către acesta.

Exemple de bootloader-e sunt:

- **GRUB** (*GRand Unified Bootloader*): bootloader-ul implicit pe majoritatea distribuţiilor Linux;
- **LILO** (*Linux Loader*): bootloader-ul implicit pe distribuţiile Linux înainte de supremaţia GRUB;
- **NTLDR** (*NT Loader*): bootloader-ul folosit de Windows NT, 2000, XP, 2003;
- **winload.exe**: bootloader-ul folosit de Windows Vista;
- **SYSLINUX**: o suită de bootloader-e destinate diverselor medii de boot.

5.2.2.1 Chainloading

Un sistem de operare poate fi boot-at direct prin specificarea imaginii nucleului către bootloader.

Alternativa o reprezintă boot-area indirectă a sistemului de operare. Acest lucru presupune ca bootloader-ului unui sistem de operare A să încarce bootloader-ul pentru sistemul de operare B, care va încărca în cele din urmă sistemul de operare B.

Bootloader-ele Linux (precum GRUB sau LILO), încarcă sistemele Windows folosind **chainloading**. Pentru aceasta, dacă după încărcarea bootloader-ului Linux (spre exemplu GRUB), se optează pentru boot-area Windows, atunci se incarcă bootloader-ul Windows (NTLDR) cu care se poate încarca sistemul de operare.

5.2.3 GRUB

GRUB (*GRand Unified Booloader*) este bootloader-ul implicit pe distribuțiile Linux şi alte sisteme de operare din familia Unix şi este, de obicei, folosit în situațiile de **multiboot** (când pe sistemul de calcul coexistă mai multe sisteme de operare). Specific GRUB este faptul că există o fază intermediară `1.5` pentru situația în care există o limită de vizibilitate a hard-disk-ului.

Ca şi în cazul altor bootloader-e, prima fază (cea care rezidă în MBR) are rolul de a încărca faza a doua sau, dacă este nevoie, faza `1.5`. Prima fază, datorită dimensiunii reduse, nu deține informații despre sistemul de fişiere. Faza `1.5`, însă, deține informații despre acesta şi poate localiza şi încărca în mod corespunzător faza a doua.

Faza a doua este responsabilă cu citirea fişierului principal de configurație (`/boot/grub/meniu.lst`), afişarea ecranului de opțiuni şi încărcarea nucleului sistemului de operare.

În figura de mai jos se prezintă localizarea diferitelor faze ale GRUB pe hard-disk:

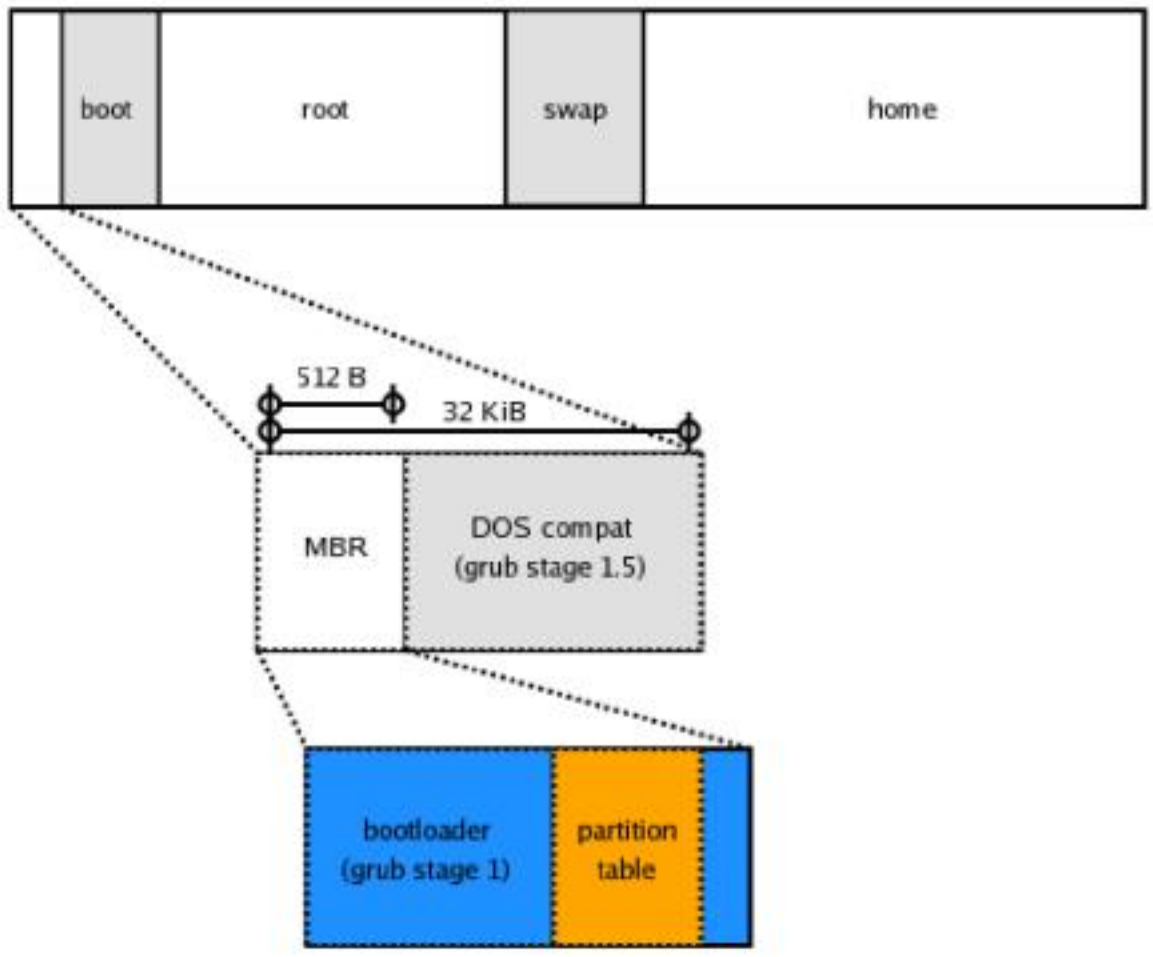


Figura 5-5 Structura GRUB

Se observă că prima fază (*grub stage 1*), se găseşte în zona de cod executabil din MBR. Faza 1.5 (*grub stage 1.5*) se găseşte în zona de compatibilitate a hard-disk-ului (primii `32 KB`). Această fază recunoaşte tipul sistemului de fişiere pe care se află faza a doua şi este utilizată pentru încărcarea acesteia. Faza a doua rezidă în partiţia de boot.

Fişierele asociate GRUB se găsesc pe un sistem Debian/Ubuntu, în `/boot/grub`, aşa cum se prezintă mai jos:

```
razvan@asgard:/boot/grub$ ls
default         fat_stage1_5  minix_stage1_5     stage2
device.map      jfs_stage1_5  reiserfs_stage1_5  xfs_stage1_5
e2fs_stage1_5   menu.lst      stage1
razvan@asgard:/boot/grub$ ls -l
total 188
-rw-r--r-- 1 root root    197 2007-01-17 11:55 default
-rw-r--r-- 1 root root     15 2007-01-17 11:55 device.map
-rw-r--r-- 1 root root   7584 2007-01-17 11:55 e2fs_stage1_5
-rw-r--r-- 1 root root   7424 2007-01-17 11:55 fat_stage1_5
-rw-r--r-- 1 root root   8192 2007-01-17 11:55 jfs_stage1_5
-rw-r--r-- 1 root root   4724 2007-06-23 12:45 menu.lst
-rw-r--r-- 1 root root   6848 2007-01-17 11:55 minix_stage1_5
-rw-r--r-- 1 root root   9280 2007-01-17 11:55 reiserfs_stage1_5
-rw-r--r-- 1 root root    512 2007-01-17 11:55 stage1
-rw-r--r-- 1 root root 108392 2007-01-17 11:55 stage2
-rw-r--r-- 1 root root   8904 2007-01-17 11:55 xfs_stage1_5
```

Se observă că prima fază se găseşte în fişierul `stage1` exact cu dimensiunea unui sector (`512`). Faza `1.5` are o formă specializată pentru fiecare sistem de fişiere (`e2fs_stage1_5`, `reiserfs_stage1_5`, etc.). Spaţiul ocupat este în jur de `7-8 KB`, deci încape fără probleme în zona de compatibilitate de `32 KB` a hard-disk-ului.

Faza a doua este descrisă de fişierul `stage2` cu dimensiune mai mare decât a celorlalte (circa `100 KB`). Faza a doua este resposabilă pentru citirea fişierului de configurare `menu.lst` şi încărcarea ulterioară a nucleului.

5.2.3.1 Vizualizare stage1 şi stage1.5 pe hard-disk

Pentru a verifica prezenţa primei faze şi a fazei `1.5` pe hard-disk vom folosi utilitarele `dd` şi `hexdump`.

În primă fază vom realiza o copie a primului sector de pe hard-disk în fişierul `first_sector` şi apoi a următoarelor `10` sectoare în fişierul `next_sectors`.

```
asgard:/boot/grub# dd if=/dev/hda of=first_sector count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 8.91e-05 seconds, 5.7 MB/s
asgard:/boot/grub# dd if=/dev/hda of=next_sectors count=10 skip=1
10+0 records in
10+0 records out
5120 bytes (5.1 kB) copied, 0.000137291 seconds, 37.3 MB/s
```

Avem acum în cele două fişiere informaţiile binare conţinute în primele sectoare ale hard-disk-ului. Vom folosi pentru comparaţie cu fişierele imagine pentru prima fază folosind utilitarul `hexdump`:

```
razvan@asgard:/boot/grub$ hexdump -C first_sector
[...]
00000070  e8 34 01 f6 c2 80 74 54  b4 41 bb aa 55 cd 13 5a  |è4.öÂ.tT´A»ªUÍ.Z|
00000080  52 72 49 81 fb 55 aa 75  43 a0 41 7c 84 c0 75 05  |RrI.ûUªuC A|.Àu.|
00000090  83 e1 01 74 37 66 8b 4c  10 be 05 7c c6 44 ff 01  |.á.t7f.L.¾.|ÆDÿ.|
000000a0  66 8b 1e 44 7c c7 04 10  00 c7 44 02 01 00 66 89  |f..D|Ç...ÇD...f.|
[...]
00000170  06 be 94 7d e8 30 00 be  99 7d e8 2a 00 eb fe 47  |.¾.}è0.¾.}è*.ëþG|
00000180  52 55 42 20 00 47 65 6f  6d 00 48 61 72 64 20 44  |RUB .Geom.Hard D|
```

```
00000190  69 73 6b 00 52 65 61 64  00 20 45 72 72 6f 72 00  |isk.Read. Error.|
000001a0  bb 01 00 b4 0e cd 10 ac  3c 00 75 f4 c3 00 00 00  |»..´.Í.¬<.uôÃ...|
[...]
razvan@asgard:/boot/grub$ hexdump -C stage1
[...]
00000070  e8 34 01 f6 c2 80 74 54  b4 41 bb aa 55 cd 13 5a  |è4.öÂ.tT´A»ªUÍ.Z|
00000080  52 72 49 81 fb 55 aa 75  43 a0 41 7c 84 c0 75 05  |RrI.ûU*uC A|.Àu.|
00000090  83 e1 01 74 37 66 8b 4c  10 be 05 7c c6 44 ff 01  |.á.t7f.L.¾.|ÆDÿ.|
000000a0  66 8b 1e 44 7c c7 04 10  00 c7 44 02 01 00 66 89  |f..D|Ç...ÇD...f.|
[...]
00000170  06 be 94 7d e8 30 00 be  99 7d e8 2a 00 eb fe 47  |.¾.}è0.¾.}è*.ëþG|
00000180  52 55 42 20 00 47 65 6f  6d 00 48 61 72 64 20 44  |RUB .Geom.Hard D|
00000190  69 73 6b 00 52 65 61 64  00 20 45 72 72 6f 72 00  |isk.Read. Error.|
000001a0  bb 01 00 b4 0e cd 10 ac  3c 00 75 f4 c3 00 00 00  |»..´.Í.¬<.uôÃ...|
[...]
```

După cum se observă, în primul sector al hard-disk-ului se găseşte imaginea primei faze, iar în următoarele sectoare imaginea fazei `1.5`. Există deosebiri între imaginea aflată în sectoarele discului şi cea din `/boot/grub` din considerente de adaptare la geometria discului. Imaginea celei de-a doua faze este încărcată de faza `1.5` din `/boot/grub/stage2`, aceasta ştiind structura sistemului de fişiere.

5.2.3.2 Creare dischetă de boot GRUB

Deşi, destul de probabil, dischetele sunt dispozitive pe cale de dispariţie (şi costă, în ziua de azi, mai mult decât un CD), vom descrie modul în care se poate crea o dischetă de boot GRUB.

Discheta va trebui să conţină imaginea pentru prima şi cea de-a doua fază (nu este nevoie de faza `1.5` întrucât nu avem limitare). Pentru aceasta putem folosi utilitarul `cat`:

```
anaconda:/boot/grub# cat stage1 stage2 > /dev/fd0
```

sau utilitarul `dd`:

```
anaconda:/boot/grub# dd if=stage1 of=/dev/fd0
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.0426716 seconds, 12.0 kB/s
anaconda:/boot/grub# dd if=stage2 of=/dev/fd0 skip=1
210+1 records in
210+1 records out
107656 bytes (108 kB) copied, 0.020834 seconds, 5.2 MB/s
```

Dacă se configurează BIOS-ul pentru a boot-a de pe dischetă, utilizatorului îi va fi oferită consola GRUB de configurare [vezi 5.2.3.4].

5.2.3.3 Configurare GRUB

Configurarea GRUB se realizează prin intermediul fişierului `/boot/grub/menu.lst`. În acest fişier sunt precizate opţiunile de boot, adică sistemele de operare care pot fi încărcate cu ajutorul GRUB. Faza a doua citeşte acest fişier de configurare şi oferă utilizatorului un ecran de opţiuni:

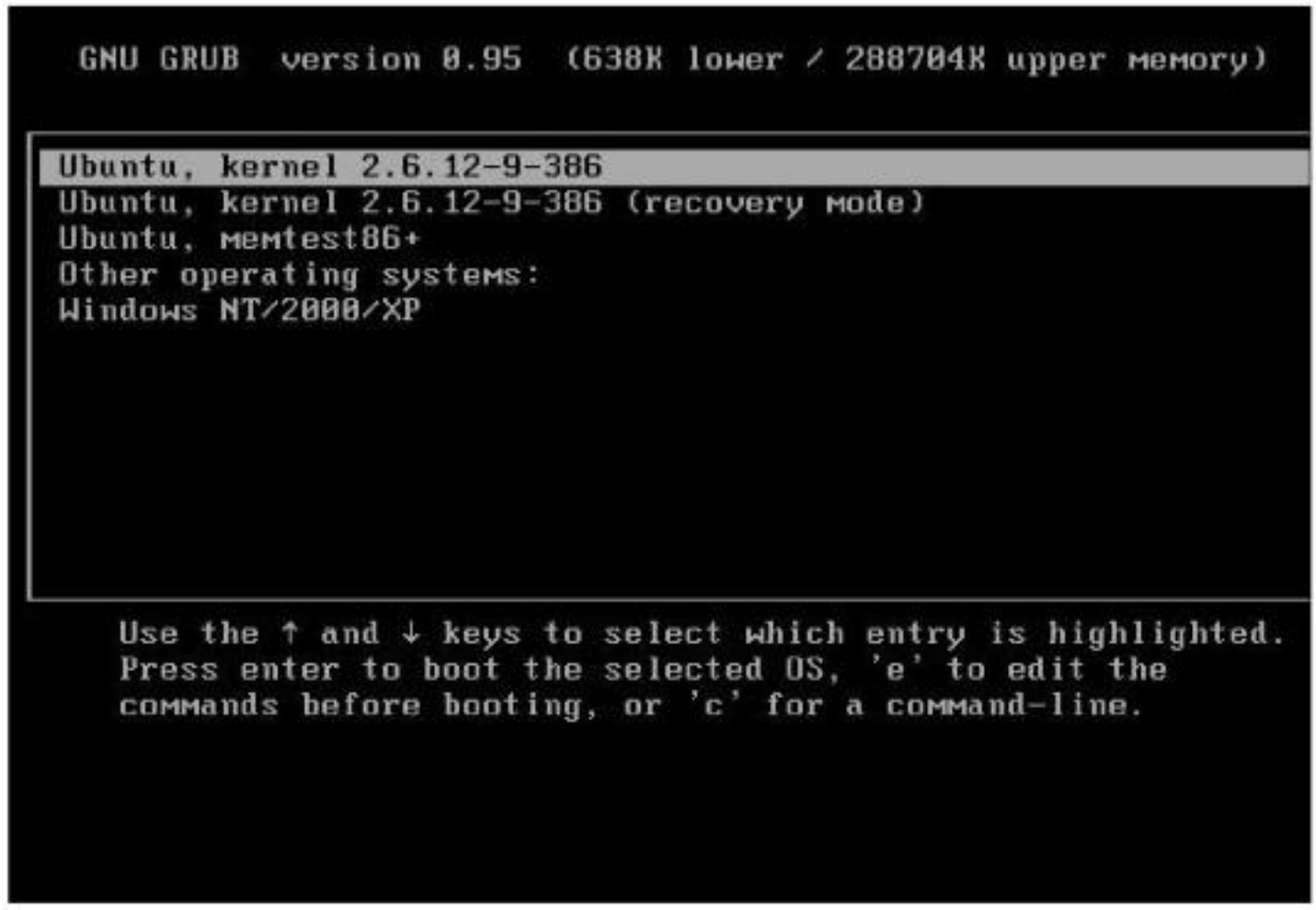


Figura 5-6 Ecran opţiuni GRUB

Prezentăm, în continuare, o parte dintr-un fişier de configurare `/boot/grub/menu.lst` şi vom descrie elementele de configurare de acolo:

```
## default num
# Set the default entry to the entry number NUM. Numbering starts from 0, and
# the entry number 0 is the default if the command is not used.
[...]
default         0

## timeout sec
# Set a timeout, in SEC seconds, before automatically booting the default entry
# (normally the first entry defined).
timeout         5

# Pretty colours
color cyan/blue white/blue

[...]

## Razvan's kernels

title           Debian GNU/Linux, kernel 2.6.18-12fast
root                    (hd0,2)
kernel          /boot/vmlinuz-2.6.18-12fast root=/dev/hda3 ro
savedefault

title           Debian GNU/Linux, kernel 2.6.18-12fast (single-user mode)
root                    (hd0,2)
kernel          /boot/vmlinuz-2.6.16-12fast root=/dev/hda3 ro single
savedefault

[...]

# This entry automatically added by the Debian installer for a non-linux OS
# on /dev/hda1
title           Microsoft Windows XP Professional
root                    (hd0,0)
savedefault
makeactive
chainloader     +1
```

Opțiunea `default` precizează opțiunea implicită folosiră pentru boot-are. Fișierul de configurare conține mai multe intrări pentru diferite sisteme de operare. Numărul care urmează după opțiunea default indică sistemul de operare care va fi boot-at implicit în cazul în care utilizatorul nu intervine în momentul afișarii ecranului de opțiuni. Numerotarea pornește de la `0`. Presupunând că, în figura de mai sus, nu ar exista alte opțiuni de sisteme de operare, boot-area implicită pe Windows XP Professional este condiționată de prezența opțiunii `default 2`.

Opțiunea `timeout` indică timpul de așteptare pentru ca utilizatorul să introducă o opțiune după afișarea ecranului de boot. Dacă timpul expiră, se încarcă sistemul de operare indicat de opțiunea default.

Opțiunea `colors` afișează colorat ecranul de opțiuni GRUB.

Opțiunea de sistem de operare

Ceea ce apare, în continuare, în exemplul de fișier de configurare de mai sus sunt intrările asociate pentru sistemele de operare care pot fi boot-ate. Se observă că fiecare intrare conține un set de directive de configurare. Acestea sunt:

- **title**: este un șir de caractere reprezentând numele intrării așa cum va apărea ea în ecranul de opțiuni GRUB; deși nu există nici o limitare de nume, se recomandă ca titlul să fie un nume reprezentativ;
- **root**: reprezintă partiția pe care se află imaginea de nucleu a sistemului de operare; aceasta poate sau nu coincide cu partiția unde este montat sistemul de fișiere rădăcină (/);
- **kernel**: reprezintă calea către imaginia de nucleu în cadrul partiției de mai sus; subdirectiva root precizează partiția asociată sistemului de fișiere rădăcină (/); opțiunea ro precizează că sistemul de fișiere va fi montat read-only; se observă că în cazul Windows nu există opțiunea kernel; aceasta deorece GRUB nu poate boot-a direct Windows ci va apela NTLDR (bootloader-ul de Windows), lucru indicat prin opțiunea `chainloader+1`; opțiunea `+1` înseamnă că se citește primul sector din partiția `(hd0,0)`;
- opțiunea **makeactive** stabilește partiția ca fiind partiția activă ;
- **savedefault**: dacă directiva default este folosită în forma default saved și opțiunea folosită la boot-are are opțiunea savedefault, următoarea boot-are va selecta, implicit, aceeași opțiune.

Convenția de denumire GRUB

Dacă în Linux, primul hard-disk este vizibil sub numele `/dev/hda`, iar prima partiție a acestuia sub numele `/dev/hda1`, GRUB folosește o altă convenție pentru denumirea discurilor din sistem și a partițiilor acestora. Astfel, pentru a referi primul disc din sistem, GRUB folosește denumire `(hd0)`. Numerotarea discurilor începe de la `0`. Spre deosebire de Linux, GRUB nu face distincție între tipul de controller folosit de discurile din sistem: IDE sau SCSI. Astfel, primul disc este denumit `(hd0)`. Pentru referirea primei partiții din primul disc, GRUB folosește denumirea `(hd0,0)`.

Dispozitiv	**Denumire Linux**	**Denumire GRUB**
Primul disc IDE	`/dev/hda`	`(hd0)`
Al doilea disc IDE	`/dev/hdb`	`(hd1)`
Al treilea disc SCSI	`/dev/sdc`	`(hd2)`
Prima unitate floppy	`/dev/fd0`	`(fd0)`
Prima partiție de pe al doilea disc IDE	`/dev/hdb1`	`(hd1,0)`
A doua partiție de pe al treilea disc SCSI	`/dev/sdc2`	`(hd2,1)`

Figura 5-7 Denumire discuri Linux/GRUB

În ton cu cele de mai sus, a 3-a partiţie de pe al doilea disc al sistemului este referită de GRUB sub forma `(hd1,2)`. Tabelul de mai sus oferă o analogie între denumirea GRUB şi Linux pentru discuri şi partiţii.

5.2.3.4 Consola GRUB

În afara fişierului de configurare GRUB poate fi configurat şi prin intermediul consolei asociate. Aceasta poate fi accesată dintr-un sistem Linux folosind comanda `grub`, sau poate fi accesată chiar din ecranul de opţiuni de boot-are. În ambele cazuri utilizatorului îi este oferit un prompt `grub>` unde poate introduce comenzi specifice.

Utilizarea consolei GRUB este limitată la situaţiile de testare a unor opţiuni, de recuperare sau de troubleshooting.

Un exemplu util de utilizare a consolei GRUB este (re)instalarea GRUB aşa cum este descrisă în secţiunea 5.5.2.1.

Altă situaţie poate apărea în momentul în care se doreşte alterarea unei opţiuni de boot-are. Spre exemplu, în ecranul de opţiuni GRUB, apăsarea tastei `e` peste o anumită opţiune conduce la apariţia consolei GRUB cu posibilitatea alterării opţiunii de bootare. Acest lucru este util în cazul unei erori (s-a precizat un nume greşit pentru imaginea de kernel) sau în cazul în care s-a pierdut parola de root şi se doreşte recuperarea acesteia, forţând sistemul să pornească o consolă de root (prin folosirea opţiunii `init=/bin/bash` [vezi 9.6.2.1]).

5.3 Încărcarea nucleului

După încărcare şi rulare, bootloader-ul este responsabil cu încărcarea nucleului sistemului de operare şi transmiterea controlului către acesta. Ca şi faza a doua a GRUB, nucleul rezidă într-un fişier din sistemul de fişiere, denumit imaginea de nucleu. În cazul GRUB, locaţia acestui fişier este transmisă ca parametru în cadrul directivei kernel.

5.3.1 Imaginea de nucleu

Imaginea de nucleu este un fişier executabil conţinând codul şi datele necesare pentru rularea nucleului sistemului de operare. În Windows, imaginea nucleului se numeşte ntoskrnl.exe şi este localizată în `C:\Windows\system32\ntoskrnl.exe`. În sistemele Linux, această imagine rezidă, de obicei, în /boot, şi are denumirea în forma `vmlinuz-versiune_kernel` (spre exemplu, `vmlinuz-2.6.21.7`).

Comanda `file` indică un astfel de fişier ca fiind imaginea de nucleu Linux:

```
  asgard:/boot# file vmlinuz-2.6.18-12fast
  vmlinuz-2.6.18-12fast: Linux kernel x86 boot executable RO-rootFS,
root_dev 0x303, swap_dev 0x1, Normal VGA
```

În mod obişnuit, imaginea de kernel ocupă în jur de `1-2 MB`. Poate fi însă constrânsă la dimensiuni mai mici prin eliminarea anumitor componente, atunci când este cazul; spre exemplu, în cazul sistemelor embedded. Faptul că denumirea imaginii se termină în `z` oferă informaţii despre faptul că imaginea este de fapt comprimată.

După încărcarea imaginii de nucleu, bootloader-ul transmite controlul execuţiei către nucleu. În primă fază nucleul iniţializează şi configurează dispozitivele hardware şi memoria. Toate dispozitivele iniţializate de BIOS vor fi reiniţializate pentru a asigura portabilitatea şi robusteţea.

Următorul pas îl reprezintă decomprimarea imaginii de nucleu. În acest pas va apărea mesajul:

```
Uncompressing Linux... Ok, booting the kernel.
```

După aceasta se montează sistemul de fişiere rădăcină precizat prin parametrul `root=/dev/...` la directiva `kernel` din fişierul de configurare GRUB. În acest moment apare mesajul:

```
VFS: mounted root (ext3 filesystem) readonly
```

Pe măsură ce nucleul iniţializează alte subsisteme, pe ecran sunt afişate alte mesaje.

Încărcarea completă a nucleului se încheie cu crearea procesului **init** (pid-ul 1) din imaginea `/sbin/init`. Se poate porni init cu altă imagine prin specificarea opţiunii `init=/path/to/new/exec` la boot-area nucleului.

5.3.2 Opţiuni de boot-are pentru nucleu

Nucleul Linux poate fi configurat pentru boot-are cu diverse argumente care să-i altereze funcţionalitatea. Acestea sunt transmise ca parametri de boot-are în directiva `kernel` a GRUB. Configurarea acestor argumente poate fi realizată fie prin fişierul de configurare (`/boot/grub/menu.lst`), fie prin accesarea consolei GRUB (apăsarea tastei e în ecranul de opţiuni GRUB).

O listă completă cu opţiunile de boot-are găsiţi la http://www.kernel.org/pub/linux/kernel/people/gregkh/lkn/lkn_pdf/ch09.pdf. Dintre opţiunile cele mai importante amintim:

- **debug**: afişează mesaje suplimentare la încărcarea nucleului;
- **quiet**: dezactivează afişarea de mesaje la încărcarea nucleului;
- **resume=suspend_device** (util pentru laptop-uri – *hibernate*): specifică discul unde se află imaginea suspendată a nucleului;
- **root=device**: precizează partiţia unde se află sistemul de fişiere rădăcină;
- **ro/rw**: sistemul de fişiere rădăcină este montat read-only/read-write la boot-are;
- **init=filename**: se specifică executabilul care va fi încărcat la crearea init (implicit acesta este `/sbin/init`);
- **S** sau **single**: sistemul va porni init în mod single-user.

5.4 Iniţializarea sistemului

Încărcarea nucleului se încheie cu pornirea **init**, procesul cu pid-ul 1, părintele celorlalte procese din sistem. Implicit, procesul init se creează din imaginea `/sbin/init`, dacă nu se foloseşte argumentul `init=/path/to/exec` la boot-area nucleului. În cele ce urmează vom presupune că procesul init a fost creat în mod obişnuit din imaginea /sbin/init.

init este responsabil cu pornirea celorlalte procese importante ale sistemului, daemonii şi a proceselor care asigură accesul utilizatorului la sistem (`getty` şi mediul grafic).

5.4.1 init

init este resposabil cu crearea celorlalte procese importante din sistem şi oferirea, astfel, către utilizator a unui sistem funcţional şi interactiv. Pentru precizarea proceselor care se doresc pornite şi a opţiunilor către acestea, init foloseşte un set de fişiere de configurare decrise în continuare.

5.4.1.1 Niveluri de rulare (runlevels)

Înainte de a prezenta modul în care se configurează init, trebuie descrisă noţiunea de nivel de rulare (runlevel). Un nivel de rulare se referă la un context de utilizare a unui sistem de operare. Altfel spus, se referă la ce servicii oferă sistemul utilizatorului. Un runlevel diferă de un altul prin dispozitivele montate şi prin serviciilor pornite.

Sunt definite `7` niveluri de rulare, numerotate de la `0` la `6`. Acestea pot fi folosite diferit de diversele distribuţii Linux, însă `3` dintre ele au tot timpul aceeaşi semnificaţie:

- Runlevel `0` – **Halt**: este folosit pentru oprirea sistemului;
- Runlevel `1` – **Single**: este folosit pentru un sistem single-user;
- Runlevel `6` – **Reboot**: este folosit pentru repornirea sistemului.

Pentru afişarea runlevel-ului curent se foloseşte comanda `runlevel` (ca root) sau `who -r` (ca utilizator obisnuit):

```
asgard:~# runlevel
N 2
razvan@asgard:~$ who -r
         run-level 2  2007-08-06 21:49                   last=S
```

Schimbarea runlevel-ului se realizează prin rularea comenzii `init` urmată de numărul runlevel-ului. Astfel rularea comenzii

```
# init 0
```

este echivalenta cu rularea comenzii:

```
# shutdown -h
```

sau

```
# halt
```

5.4.1.2 Configurarea init

/etc/inittab

Fişierul principal de configurare pentru init este `/etc/inittab`. Prezentăm, în continuare, o parte a unui fişier de configurare `/etc/inittab` pentru a descrie modul în care acesta afectează comportamentul init:

```
# The default runlevel.
id:2:initdefault:
[...]
# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
[...]

# /sbin/getty invocations for the runlevels.
#
```

```
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
#
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

După cum se observă intrările din `/etc/inittab` au formatul:

```
<id>:<runlevels>:<action>:<process>
```

unde:

- **id** este un identificator unic al regulii din inittab;
- **runlevels** reprezintă nivelurile de rulare asupra cărora se aplică regula;
- **action** este acțiunea pe care o va executa init;
- **process** este comanda care va fi executată.

O descriere completă a regulilor posibile pentru inittab se găseşte în pagina de manual (`man inittab`). Vom prezenta, în continuare, regulile din exemplul de mai sus.

Prima linie de configurare din fişier stabileşte nivelul implicit de rulare care este `2`.

Identificatorii `10` până la `16` prezintă reguli pentru scripturile de iniţializare a serviciilor sistemului conform cu nivelul de rulare. După cum se observă, comanda de executat pentru fiecare runlevel în parte este `/etc/init.d/rc` urmată de numărul nivelului de rulare. Prin intermediul acestei comenzi se vor porni serviciile specifice sistemului, după cum se va vedea în continuare.

Linia cu identificatorul ca precizează că se execută o repornire a sistemului în cazul în care se apasă `CTRL-ALT-DEL`.

Regulile cu identificatorii `1` până la `6` sunt folosite pentru a porni procesul getty pe terminalele virtuale ale sistemului. După cum se observă şi din comentariu, terminalul virtual `7` este asociat mediului grafic şi nu poate fi folosit pentru a porni `getty`. Acţiunea respawn înseamnă că în cazul terminării procesului, init îl va recrea.

5.4.1.3 Pornirea daemonilor de sistem

După cum s-a observat din fişierul de configurare `/etc/inittab`, pornirea serviciilor sistemului în funcţie de runlevel se realizează cu ajutorul scriptului `/etc/init.d/rc`. Acest script primeşte ca argument numărul runlevel-ului şi execută scripturile dintr-un director specializat: `/etc/rcX.d/`, unde `X` este numărul runlevel-ului. Astfel, dacă runlevel-ul este `3` se vor executa script-urile din `/etc/rc3.d/`.

Să aruncăm o scurtă privire peste `/etc/rc3.d/`:

```
razvan@asgard:~$ ls -l /etc/rc3.d/
total 4
lrwxrwxrwx 1 root   root    18 2007-06-10 22:56 K08vmware -> /etc/init.d/vmware
lrwxrwxrwx 1 root   root    20 2007-07-22 14:17 K77ntp-server -> ../init.d/ntp-server
[...]
lrwxrwxrwx 1 razvan razvan  15 2007-01-17 11:54 S20exim4 -> ../init.d/exim4
lrwxrwxrwx 1 razvan razvan  19 2007-06-01 12:51 S20firebird2 -> ../init.d/firebird2
[...]
lrwxrwxrwx 1 root   root    18 2007-06-10 22:56 S90vmware -> /etc/init.d/vmware
lrwxrwxrwx 1 razvan razvan  17 2007-02-24 00:05 S91apache2 -> ../init.d/apache2
lrwxrwxrwx 1 razvan razvan  18 2007-01-17 11:47 S99rc.local -> ../init.d/rc.local
lrwxrwxrwx 1 razvan razvan  19 2007-01-17 11:47 S99rmnologin -> ../init.d/rmnologin
lrwxrwxrwx 1 razvan razvan  23 2007-01-17 11:47 S99stop-bootlogd -> ../init.d/stop-bootlogd
```

Se observă că avem un set de legaturi simbolice către intrări din `/etc/init.d/`. Directorul `/etc/init.d/` conţine scripturile de pornire/oprire/repornire a serviciilor din sistem. Astfel, pentru oprirea, pornirea sau repornirea serviciului apache se folosesc comenzile:

```
# /etc/init.d/apache start|stop|restart
```

Argumentele start/stop/restart sunt comune majorităţii scripturilor din `/etc/init.d`.

De asemenea, se observă un format al intrărilor din `/etc/rcX.d`:

```
K număr nume
S număr nume
```

`K` înseamnă `kill`, iar `S` înseamnă `start`. Astfel, existenţa legăturii `K08vmware` înseamnă că serviciul `vmware` va fi oprit o dată cu intrarea în nivelul de rulare `3` (prin intermediul comenzii `/etc/init.d/vmware stop`). La fel, existenţa legăturii `S20exim4` înseamnă că serviciul `exim` va fi pornit o dată cu intrarea în nivelul de rulare `3` (prin intermediul comenzii `/etc/init.d/exim start`).

Numărul asociat reprezintă ordinea de pornire/oprire a serviciilor. Astfel, serviciul `S91apache` va fi pornit înaintea `S99rmnologin`, iar serviciul `K77ntp-server` va fi oprit înaintea `K08vmware` (descrescător la oprire).

sysv-rc-conf

Avantajul schemei init de pornire a serviciilor este flexibilitatea cu dezavantajul complexităţii şi a unui număr mare de fişiere de configurare. Dacă dorim eliminarea sau adăugarea unui serviciu la un runlevel va trebui să creăm o legătură simbolică către scriptul de pornire/oprire şi să-i dăm un nume specific. Procesul poate fi anevoios în cazul în care dorim o configurare mai avansată.

Pentru a elimina acest deficient, se poate folosi utilitarul `sysv-rc-conf`. Acesta permite adăugarea/eliminarea facilă a unui serviciu dintr-un runlevel.

`sysv-rc-conf` are o interfaţă în linia de comandă şi una text pentru configurarea serviciilor. Interfaţa text permite parcurgerea serviciilor existente în sistem şi adăugarea/elimitarea acestora la un runlevel printr-o simplă acţiune de *check/uncheck* (folosind tasta `SPACE`). Interfaţa în linia de comandă permite adăugarea/eliminarea unui serviciu prin precizarea nivelului/nivelurilor de rulare dorite, a numelui serviciului şi a acţiunii (on sau off). În exemplul de mai jos, am adăugat serviciul `postfix` la nivelurile de rulare `4` şi `5` şi am eliminat serviciul `apache` din nivelurile de rulare `2` şi `5`.

```
anaconda:/boot/grub# sysv-rc-conf --list postfix
postfix      0:off     1:off   2:on    3:off   4:off   5:off   6:off
anaconda:/boot/grub# sysv-rc-conf --level 45 postfix on
anaconda:/boot/grub# sysv-rc-conf --list postfix
postfix      0:off     1:off   2:on    3:off   4:on    5:on    6:off

anaconda:/boot/grub# sysv-rc-conf --list apache
apache       0:off     1:off   2:on    3:on    4:on    5:on    6:off
anaconda:/boot/grub# sysv-rc-conf --level 25 apache off
anaconda:/boot/grub# sysv-rc-conf --list apache
apache       0:off     1:off   2:off   3:on    4:on    5:off   6:off
```

5.4.2 upstart

Începând cu versiunea 6.10 (Edgy Eft), distribuţia Ubuntu a înlocuit init cu **upstart**. upstart permite o configurare asincronă, bazată pe evenimente care au loc în sistem. Spre deosebire de alternative pentru init, upstart este compatibil cu acesta şi poate folosi nemodificate scripturile

acestuia (spre exemplu cele din `/etc/init.d`). upstart este, de fapt, un daemon init modificat. Ca orice sistem Linux, este pornit prin rularea `/sbin/init`.

`upstart` a apărut ca urmare a dezvoltării hardware-ului. Dacă până acum se putea şti din momentul pornirii sistemului ce dispozitive există în sistem, unde putem monta sistemul de fişiere, dezvoltarea de noi dispozitive *hot-plug* (care pot fi conectate sau deconectate în timp real în calculator) a impus folosirea unui nou sistem de detecţie şi configurare a acestora.

`upstart` este aşadar un daemon init bazat pe evenimente (*event-based init daemon*). Funcţionarea sa înseamnă crearea de noi sarcini pentru sistemul de operare în două moduri:

- rularea de scripturi sau executabile în momentul producerii unui eveniment;
- oprirea sau pornirea unui serviciu în momentul producerii unui eveniment.

Un exemplu este montarea unui sistem de fişiere (spre exemplu de pe un USB flash drive). Paşii urmaţi sunt:

- un eveniment tip *startup* porneşte daemon-ul udevd;
- daemon-ul udevd este configurat să trimită un eveniment de tipul *block-device-added* pentru fiecare dispozitiv;
- evenimentul *block-device-added* determină verificarea dispozitivului şi montarea acestuia dacă este listat în `/etc/fstab`;
- după montarea sistemelor de fişiere din `/etc/fstab` se transmite evenimentul *fhs-filesystem*.

5.4.2.1 Configurare upstart

`upstart` este configurat prin intermediul directorului `/etc/event.d/` care se doreşte a fi un înlocuitor pentru scripturile `/etc/inittab`, `/etc/init.d/*` şi `/etc/rcX.d`. Fiecare fişier din `/etc/event.d` descrie modul în care un job trebuie coordonat:

```
root@ubuntui:/etc/event.d# ls
control-alt-delete  rc0  rc3  rc6          sulogin  tty3  tty6
logd                rc1  rc4  rcS          tty1     tty4
rc-default          rc2  rc5  rcS-sulogin  tty2     tty5
```

Fişierul `control-alt-delete` prezintă acţiunea întreprinsă la apăsarea `CTRL-ALT-DEL`, anume repornirea sistemului:

```
root@ubuntui:/etc/event.d# cat control-alt-delete
# control-alt-delete - emergency keypress handling
#
# This task is run whenever the Control-Alt-Delete key combination is
# pressed.  Usually used to shut down the machine.

start on control-alt-delete

exec /sbin/shutdown -r now "Control-Alt-Delete pressed"
```

Scripturile `rc*` sunt aociate cu nivelurile de rulare. După cum se observă, la fel ca în `/etc/inittab`, se apelează `/etc/init.d/rc` cu argument numărul nivelului de rulare:

```
root@ubuntui:/etc/event.d# cat rc2
[..]
start on runlevel 2
stop on runlevel [!2]
script
[...]
        exec /etc/init.d/rc 2
end script
```

Scripturile `tty*` prezintă modul de pornire a getty pentru diversele terminale virtuale (la fel ca în `/etc/inittab`).

Începând cu Ubuntu 7.04 (Feisty Fawn), upstart a înlocuit câteva procese daemon importante cum sunt `cron`, `atd`, `anacron` şi `inetd`.

5.4.3 getty şi login

După pornirea daemonilor sistemului, init este responsabil cu pornirea proceselor care asigură autentificarea utilizatorului la sistemul de calcul. După cum s-a văzut în cadrul fişierului de configurare `/etc/inittab`, init porneşte câte un proces **getty** pentru fiecare terminal virtual; getty porneşte din imaginea `/sbin/getty`. În mod implicit sunt pornite `6` procese `getty` pentru terminalele denumite `tty1`, `tty2`, ..., `tty6`. Terminalul `tty7` este rezervat mediului grafic (în cazul în care acesta este instalat).

Terminalele virtuale sunt accesate prin intermediul combinaţiei de taste `CTRL-ALT-Fn`, unde `n` este numărul terminalului. Combinaţia `CTRL-ALT-F7` este folosită pentru acccesarea mediului grafic.

`getty` este responsabil cu crearea unei interfeţe de autentificare a utilizatorului. `getty` este folosit atât pentru autentificarea pe terminale virtuale cât şi pentru conexiunea pe alte terminale precum interfeţele seriale.

După iniţializarea terminalului, `getty` oferă prompt-ul `"Login:"` către utilizator pe terminalul virtual. Utilizatorul va trebui să introducă numele de utilizator şi să apese `ENTER`.

După introducerea numelul de utilizator, getty invocă `/bin/login`. Astfel, imaginea sa este înlocuită cu imaginea `/bin/login` şi se creează procesul **login**.

În continuare este cerută parola utilizatorului. În acest moment, login poate verifica validitatea utilizatorului care a dorit să se autentifice. Dacă utilizatorul a introdus o pereche nume de utilizator/parolă corectă atunci, în mod tipic, i se va deschide o sesiune de shell pe terminalul curent. Altfel, procesul login moare, şi, după un interval de aşteptare, getty este repornit şi oferă, din nou, promptul de autentificare utilizatorului.

Schema de autentificare şi pornire a sesiunii de shell (în cazul unei autentificări reuşite), este prezentată în figurile de mai jos:

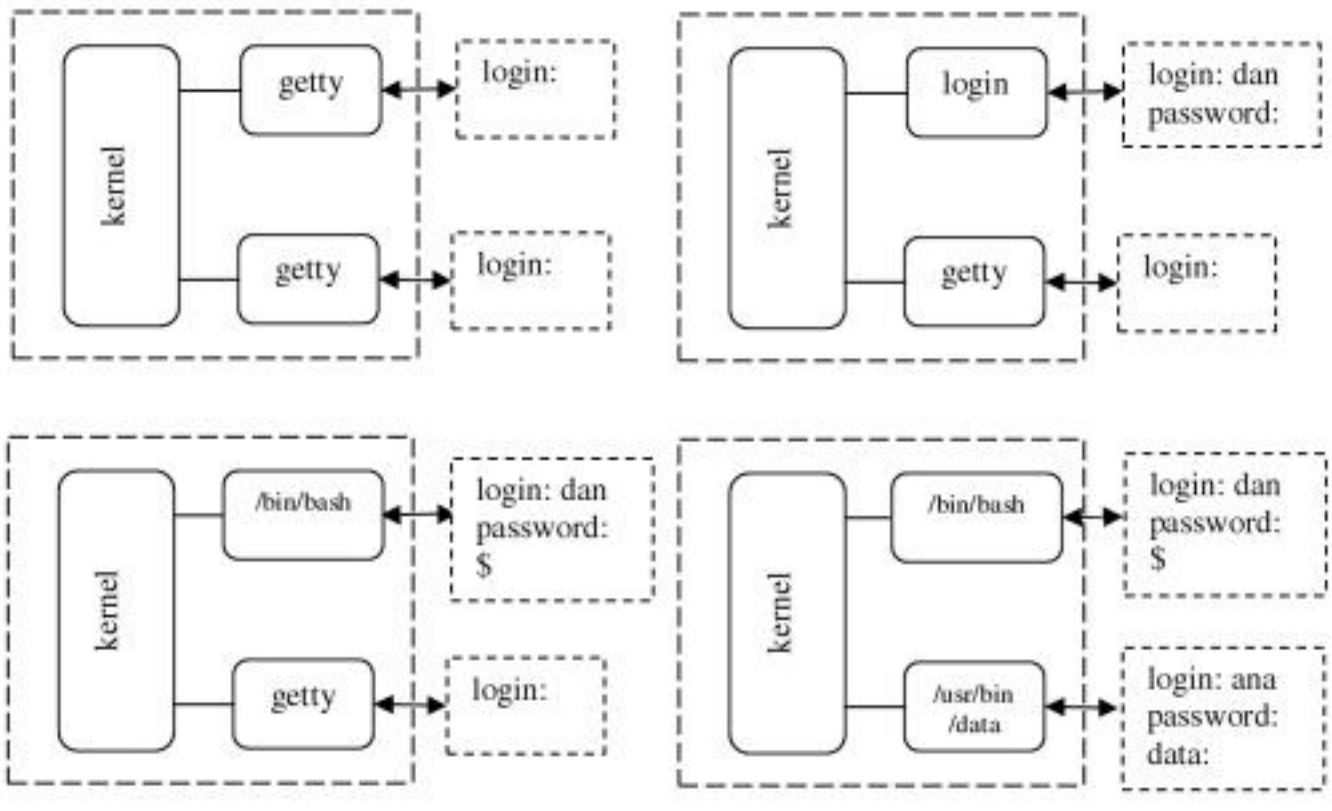


Figura 5-8 Autentificarea într-un sistem Linux

Se observă că se rulează getty pe toate terminalele. În momentul în care utilizatorul introduce numele de utilizator, getty dă naştere procesului login care cere parola utilizatorului.

O dată cu introducerea şi validarea parolei, utilizatorului îi este permis accesul în sistem şi se deschide, în mod tipic, o sesiune shell. După cum se observă în ultima imagine, utilizatorul sue rulează shellul Bourne (`sh - /usr/bin/sh`), pat rulează shellul Korn (`ksh`), iar bob rulează o comandă proprie.

Vom vedea că aceste comenzi sunt date de configuraţia existentă în fişierul `/etc/passwd`.

La încheierea sesiunii de shell curente, procesul getty este reponit aşa cum este configurat în `/etc/inittab` (`respawn`).

5.4.3.1 Fişierele /etc/passwd şi /etc/shadow

Fişierele `/etc/passwd` şi `/etc/shadow` sunt utilizate pentru autentificarea utilizatorilor în sistem şi oferirea de informaţii despre acestea. După com se observă dintr-o analiză simplă a fişierului `/etc/passwd`:

```
razvan@asgard:~$ cat /etc/passwd | grep guest
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
```

o intrare a acestuia este de forma:

```
username:simbol_parola:uid:gid:extra_info:home_dir:interpretor
```

Vom insista pe două intrări care sunt folosite de login pentru autentificarea utilizatorilor: **simbol_parola** şi **interpretor**.

Câmpul interpretor specifică ce comandă va fi executată în momentul în care un utilizator a fost autentificat cu succes în sistem. De obicei, această comandă este un interpretor de comenzi (în exemplul nostru `/bin/bash`), dar poate fi configurată orice altă comandă. În imaginea secţiunii anterioare, utilizatorul bob folosea `/usr/data/bin/data_entry`. Dacă veţi completa intrarea asociată cu `/usr/bin/vim`, atunci orice autentificare pe un terminal virtual va rezulta în deschiderea editorului vim pe acel terminal.

Al doilea câmp dintr-o intrare în `/etc/passwd` (denumit aici simbol_parola) era folosit iniţial pentru a stoca parola criptată. În felul acesta, oricine avea acces la fişier avea acces la parola criptată şi putea folosi diverşi algoritmi pentru a o sparge. O soluţie ar fi fost restricţionarea accesului la fişier. Totuşi, acesta conţine şi alte informaţii necesare altor comenzi. Soluţia aleasă a fost crearea unui nou fişier `/etc/shadow` care poate fi accesat doar de utilizatorul privilegiat.

```
anaconda:~# ls -l /etc/shadow
-rw-r----- 1 root shadow 3419 Aug  5 18:37 /etc/shadow
```

Fişierul shadow conţine numele de utilizator şi parola criptată. Pentru mai multe detalii consultaţi paginile de manual (`man 5 passwd`, `man 5 shadow`) sau capitolul de securitate [vezi 9.2.2].

5.4.4 Sesiune de shell

După rularea init şi autentificarea utilizatorului în sistem acestuia îi este, în mod tipic, oferită o sesiune de shell (conform configuraţiei din fişierul `/etc/passwd`).

Înaintea oferirii promptului către utilizator, procesul shell nou creat citeşte un set de fişiere de configurare. Acestea sunt folosite pentru a stabili formatul promptului, umask-ul curent, aliasurile de comenzi, variabilele de mediu.

În cazul unui interpretor bash (*Bourne Again Shell*), fişierele de configurare citite sunt în ordine:

- `/etc/profile`: acesta este fişierul global de configurare pentru sesiunile de shell;
- `~/.bash_profile` sau `~/.bash_login` sau `~/.profile`: se execută primul găsit;

- `~/.bashrc` este fişierul obişnuit de configurare locală; drept urmare, `~/.bash_profile` conţine linia

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

pentru a interpreta acest fişier.

După interpretarea fişierelor de configurare, se stabileşte mediul curent de lucru şi utilizatorului îi este oferit promptul unde poate introduce comenzi.

5.5 Studiu de caz

5.5.1 Pornirea sistemului de operare Windows

Pornirea sistemului de operare Windows urmăreşte aceeaşi paşi ca în cazul pornirii Linux: pornire sistem, încărcare BIOS, încărcare bootloader, încărcare nucleu, pornire servicii.

Paşii până la încărcarea bootloader-ului sunt independenţi de sistemul de operare şi, deci, identici cu cei pentru Linux.

Bootloader-ul pentru Windows NT (NT, 2000, XP, 2003) este **NT Loader** (NTLDR).

5.5.1.1 NTLoader

La fel ca şi în cazul Linux, sectorul de boot al hard-disk-ului (MBR) sau al unei partiţii (VBR) este limitat şi este folosit pentru încărcarea NTLDR. Imaginea acestuia rezidă în `C:\Windows\ntldr`.

După încărcarea bootloader-ului, acesta consultă fişierul boot.ini pentru a localiza informaţiile de pe partiţia de sistem.

Configurare NTLoader (boot.ini)

Fişierul `boot.ini` se află pe partiţia de sistem. Pentru vizualizarea şi accesarea acestuia trebuie permisă vizualizarea fişierelor de sistem (`Tools->Folder Options->View->Hide protected operating system files`).

Un exemplu de conţinut al `boot.ini` este prezentat în continuare:

```
  [boot loader]
  timeout=30
  default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
  [operating systems]
  multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP
Professional" /noexecute=optin /fastdetect
```

În cazul existenţei mai multor sisteme Windows, utilizatorului îi va fi afişat un ecran de selecţie (asemănător cu cel al GRUB).

Variabila timeout specifică timpul de aşteptare înainte de a încărca sistemul de operare dat de opţiunea implicită.

Puteţi consulta lista cu opţiunile posibile de boot-are a nucleului Windows aici: http://support.microsoft.com/default.aspx?scid=kb;en-us;833721.

5.5.1.2 Încărcare nucleu

După rularea bootloader-ului, acesta încarcă în memorie imaginea de nucleu `ntoskrnl.exe` şi `hall.dll` (*Hardware Abstraction Layer*). După încărcarea acestora, controlul este oferit nuclelului.

Nucleul este încărcat în memorie în două faze: în prima fază se iniţializează dispozitivele sistemului (procesor, memorie), iar în a doua fază se încarcă device driver-ele de sistem necesare.

5.5.1.3 Pornire procese importante

După încărcarea nucleului în memorie, sunt pornite, în ordine, două procese importante:

- **smss** (*Session Manager Subsystem*): acesta montează dispozitivele din sistem, creează variabilele de mediu şi porneşte winlogon (*Windows Logon Manager*);
- **winlogon** (*Windows Logon Manager*): este procesul care se ocupă de autentificarea utilizatorilor în sistem; în urma unei autentificări reuşite sunt pornite shell-ul Windows (`explorer`) şi alte procese ale utilizatorului.

Mai multe detalii despre procesul de pornire a unui sistem Windows găsiţi aici: http://technet.microsoft.com/en-us/library/bb457123.aspx.

5.5.2 Interoperabilitate Linux-Windows

În cazul sistemelor dual-boot se poate întampla să se suprascrie MBR-ul şi să se piardă astfel opţiunea de boot-are a unui sistem de operare. Pentru aceasta trebuie reconfigurat MBR-ul pentru a putea încărca bootloader-ul necesar.

5.5.2.1 Reinstalare GRUB

Situaţia cea mai frecventă este aceea în care se reinstalează Windows-ul pe un sistem pe care există instalată şi o distribuţie Linux. Din nefericire, la instalarea Windows-ului, acesta suprascrie MBR astfel încât se pierde stage1 din GRUB.

Pentru a reinstala GRUB, avem nevoie de un mediu bootabil Linux. Acesta poate fi un LiveCD sau o dischetă boot-abilă. În cazul unei dischete boot-abile utilizatorului îi este oferit direct prompt-ul de grub. În cazul unui LiveCD, utilizatorul va trebui să deschidă o consolă şi să acceseze de acolo consola de GRUB:

```
    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

       [ Minimal BASH-like line editing is supported.  For
         the  first  word,  TAB  lists  possible  command
         completions.  Anywhere else TAB lists the possible
         completions of a device/filename. ]

grub>
```

Presupunând că vrem să instalăm GRUB pe primul hard-disk din sistem, va trebui să urmăm paşii de mai jos:

- configurarea partiţiei care conţine imaginile de GRUB:

```
grub> root (hd0,2)
```

- dacă nu se ştie pe ce partiţie se află imaginile se poate folosi comanda `find`:

```
grub> find /boot/grub/stage1
 (hd0,2)
```

- după descoperirea partiţiei se instalează imaginea de GRUB în MBR:

```
grub> setup (hd0)
```

- imaginea de GRUB poate fi instalată şi în sectorul de boot al unei partiţii (VBR):

```
grub> setup (hd0,0)
```

însă, în acest caz, GRUB va trebui să fie pornit (*chainloaded*) cu un alt bootloader.

Comanda `quit` înseamnă ieşire din consola GRUB. După repornirea sistemului, acesta va afişa ecranul de opţiuni GRUB pentru selecţia sistemului de operare dorit.

5.5.2.2 Reinstalare NTLoader

Reinstalarea NTLDR este necesară în cazul în care se instalează o distribuţie Linux fără a se instala şi un bootloader (spre exemplu GRUB), sau în cazul unui MBR corupt.

Petru repararea MBR, trebuie folosită consola de recuperare Windows (*Recovery Console*). Pentru accesarea acesteia se foloseşte CD-ul de instalare Windows, şi se alege opţiunea *Repair*. În continuare se apasă `R` pentru rularea *Emergency Repair Process* şi se alege *Fast* sau *Manual Repair*. *Fast Repair* este recomandată majorităţii utilizatorilor, iar *Manual Repair* administratorilor şi utilizatorilor avansaţi.

Mai multe detalii găsiţi aici: http://www.ntfs.com/missing-corrupted-system-files.htm

5.5.3 Personalizare ecran GRUB

Cei plictisiţi de aceeaşi ecran de opţiuni prezentat de GRUB pot folosi o altă imagine de fundal. Această imaginea se numeşte GRUB *splashimage*. Există numeroase site-uri care oferă astfel de imagini (spre exemplu: http://schragehome.de/splash/,. http://www.queervisions.com/arch/2007/04/9_grub_splash_i.html).Imaginile sunt imagini `.xpm` arhivate (de exemplu `hubble.xpm.gz`).

Pentru folosirea unei imagini *splash* de GRUB trebuie editat fişierul principal de configurare a GRUB (`/boot/grub/menu.lst`). Paşii de urmat sunt prezentaţi mai jos:

- crearea unei copii de siguranţă a fişierului de configurare:

```
asgard:/boot/grub# cp menu.lst menu.lst.old
```

- obţinerea unui fişier `.xpm.gz` care conţine imaginea de *splash* folosită pentru GRUB:

```
asgard:/boot/grub# wget http://schragehome.de/splash/hubble.xpm.gz
--14:39:24--  http://schragehome.de/splash/hubble.xpm.gz
           => `hubble.xpm.gz.1'
Resolving schragehome.de... 82.165.105.215
Connecting to schragehome.de|82.165.105.215|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 99,593 (97K) [image/x-xpixmap]

100%[====================================>] 99,593        283.93K/s

14:39:25 (283.15 KB/s) - `hubble.xpm.gz.1' saved [99593/99593]
```

- editarea fişierului de configurare:

```
[...]
# Pretty colours
#color cyan/blue white/blue
splashimage=(hd0,2)/boot/grub/hubble.xpm.gz
[...]
```

- în exemplul de mai sus am folosit faptul că partiţia unde se află imaginea de GRUB este a treia partiţie de pe primul disc `(hd0,2)`.

După editarea şi salvarea fişierului se reporneşte sistemul. Noul ecran de opţiuni GRUB va folosi ca fundal imaginea *splash* indicată în fişierul de configurare.

Cuvinte cheie

- pornirea/repornirea unui sistem
- booting/bootstrapping
- fazele pornirii unui sistem de operare
- BIOS
- POST
- CMOS
- dispozitiv boot-abil
- sector de boot/sector bootabil
- MBR, VBR
- tabelă de partiții
- bootloader
- fazele unui bootloader (stages)
- GRUB
- stage1, *_stage1_5, stage2
- /boot/grub/menu.lst
- consola GRUB
- imagine de nucleu
- opțiuni de încărcare a nucleului
- init
- runlevel
- /etc/inittab
- /etc/init.d, /etc/rcX.d
- getty
- login
- /etc/passwd, /etc/shadow
- /etc/prfile, ~/.bash_profile, ~/.bashrc
- NTLDR
- boot.ini
- smss, winlogon
- Recovery Console

Întrebări

Care este ordinea corectă a pornirii unui sistem?

- ❑ vmlinuz, init, GRUB, BIOS
- ❑ GRUB, BIOS, init, vmlinuz,
- ❑ BIOS, GRUB, vmlinuz, init
- ❑ init, vmlinuz, BIOS, GRUB

Ce este un sector de boot?

- ❑ primul sector al primei partiții
- ❑ primul sector al unei partiții sau al unui hard-disk
- ❑ un sector ce conține semnătura 0xAA55 (ultimii doi octeți)
- ❑ un sector pe care se găseşte faza a doua a GRUB (stage2)

Ce este un sector bootabil?
- primul sector al primei partiţii
- primul sector al unei partiţii sau al unui hard-disk
- un sector ce conţine semnătura 0xAA55 (ultimii doi octeţi)
- un sector pe care se găseşte faza a doua a GRUB (stage2)

Câte partiţii primare, active, extinse, logice se pot afla pe un hard-disk?
- oricâte, 1, 2, 4
- 1, oricâte, oricâte, 4
- 4, 1, 1, oricâte
- 1, oricâte, 2, oricâte

Care din următoarele asocieri nu este validă?
- GRUB/NTLDR
- vmlinuz/ntoskrnl
- POST/CMOS
- getty/winlogon

Care fază a GRUB este resposabilă cu încărcarea imaginii de nucleu Linux?
- stage1
- stage1.5
- stage2
- stage3

Care este fişierul principal de configurare GRUB pe un sistem Debian/Ubuntu?
- /etc/main.cf
- /etc/grub.conf
- /boot/grub.conf
- /boot/grub/meniu.lst

Care este ordinea corectă de pornire a unei sesiuni shell?
- login, init, getty, bash
- getty,bash, login, init
- init, getty, login, bash
- init, login, bash, getty

Cum se reprezintă, în notaţia GRUB, a 3-a partiţie de pe al doilea disc?
- /dev/hdb3
- /dev/hda2
- (hd1,2)
- (hd2,3)

Fişierul /etc/shadow conţine parolele criptate ale utilizatorilor sistemului. Poate fi accesat numai de root.
- adevărat/adevărat
- adevărat/fals
- fals/adevărat
- fals/fals

6 Analiza hardware a sistemului

WARNING: Keyboard Not Attached. Press F10 to Continue

Ce se învaţă din acest capitol

- Principalele componente hardware ale unui sistem de calcul
- Mecanismele de accesare şi interogare a dispozitivelor hardware
- Încărcarea şi folosirea modulelor de kernel
- Utilitare de interacţiune cu dispozitivele: lspci, lsusb, smartmontools
- Lucrul cu dispozitivele sistemului; comanda dd
- Interacţiunea cu sistemul: /proc, sysctl
- Cum se poate realiza backup la MBR

6.1 Structura unui sistem de operare

Sistemul de operare este cel care se ocupă de managementul resurselor hardware şi software prezente în computer. El are mai multe componente, fiecare cu un rol bine definit. Diagrama următoare prezintă modul în care este structurat accesul la hardware:

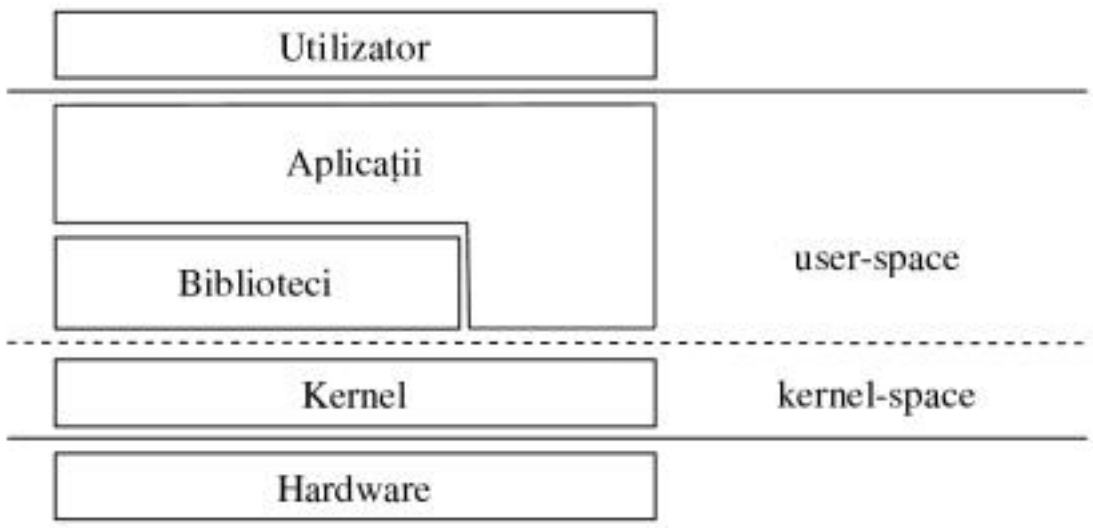


Figura 6-1 – Structura unui sistem de operare

Sistemul de operare este compus din:

- **Kernel**: nucleul sistemului de operare - se ocupă de comunicaţia directă cu hardware-ul şi oferă un sistem de management al resurselor pentru aplicaţiile din sistem

- Biblioteci de funcții - pun la dispoziție aplicațiilor un API (Application Programming Interface) comun pentru accesul la resursele puse la dispoziție de kernel
- Aplicații – aplicațiile care vin cu un sistem de operare sunt de obicei utilizate pentru configurarea sistemului, modificarea datelor din sistem etc. Aceste aplicații pot să utilizeze biblioteci de funcții sau pot comunica direct cu kernel-ul.

Sistemul de operare face distincția între două „spații" în care se pot găsi programe in execuție: spațiul utilizator (*user-space*) şi spațiul kernel (*kernel-space*). Prin „spațiu" se înțelege atât spațiu de memorie cât si mediu de execuție. Fiecare spațiu are alte condiții de lucru.

***Spațiul kernel** este spațiul în care rulează kernel-ul. Conține doar kernel-ul si componentele sale (precum driver-ele încărcate la un moment dat). Tot ceea ce se găseşte în spațiul kernel are acces direct la hardware.*

***Spațiul utilizator** este spațiul în care se găsesc aplicațiile şi bibliotecile de funcții utilizate de acestea la un moment dat. Tot ceea ce se găseşte în spațiul utilizator **nu** are acces direct la hardware.*

Toate acțiunile aplicațiilor din spațiul utilizator trebuie să treacă prin kernel. Comunicația dintre spațiul utilizator şi spațiul kernel se realizează prin apeluri de sistem.

***Apelul de sistem** este un mecanism prin care programele sau bibliotecile din spațiul utilizator accesează resursele puse la dispoziție de kernel.*

Practic, apelurile de sistem sunt funcții implementate în kernel pe care aplicațiile din spațiul utilizator le accesează direct sau prin intermediul bibliotecilor de funcții.

În prezent, majoritatea sistemelor de operare vin cu suport şi pentru alte platforme, platforme hardware (alte procesoare) sau software (alt format pentru aplicații). Suportul pentru aceste platforme este inclus prin intermediul maşinilor virtuale.

***Maşina virtuală** este un sistem care creează un mediu virtual pentru o anumită platformă într-un sistem de operare, astfel încât să se poată executa aplicații pe maşina virtuală precum pe o maşina reală.*

Maşinile virtuale care simulează platforme hardware au fost menționate în secțiunea 1.4.2. Pe lângă acestea, există şi maşini virtuale care permit execuția unor aplicații care nu au fost create pentru un anumit sistem de operare/platformă.

Cel mai elocvent exemplu este *maşina virtuală Java (JVM[14])* care poate executa aplicații Java. Aceste aplicații nu sunt ca aplicațiile normale compilate în cod maşină. Aplicațiile Java sunt compilate în Java-bytecode iar acest cod poate fi executat pe orice sistem care are o maşină virtuală Java.

Un alt mod de a rula aplicații pentru alte platforme este prin intermediul unui translator dinamic de cod binar. La execuția unui program se iau bucăți de cod binar şi se traduc în cod binar pentru maşina care execută codul. Rosetta[15] este un astfel de translator inclus în Mac OS X pentru a permite execuția aplicațiilor compilate pentru procesoare Power PC pe procesoare Intel (pentru sistemul de operare Mac OS).

Acest capitol va analiza modul de lucru intern pentru un sistem Linux.

---

[14] *JVM – Java Virtual Machine: cea mai cunoscută maşina virtuală Java este cea de la Sun (http://java.sun.com); mai există maşini virtuale Java şi de la alți producători, precum Microsoft*

[15] *Rosetta - http://www.apple.com/rosetta/*

6.2 Considerente hardware

Componentele fizice ale sistemului poartă denumirea generică de hardware. Pentru folosirea acestora, utilizatorul dispune de un sistem de operare şi aplicaţii. Acestea reprezintă nivelul de software de peste hardware. Fiecare componentă este esenţială şi nu poate funcţiona în lipsa celeilalte.

Sistemul de operare şi aplicaţiile de bază sunt cele care asigură interfaţa cu hardware-ul pentru a preveni, astfel, dificultatea programării directe a hardware-ului. Din punct de vedere al utilizatorului la programatorului, interfaţa cu hardware-ul este limitată de un set de apeluri către sistemul de operare şi aplicaţiile de bază.

6.2.1 Structura unui sistem de calcul

Din punct de vedere structural, un sistem de calcul este un set de componente cu roluri bine definite care comunică pentru îndeplinirea unei sarcini. Sistemele de calcul din ziua de azi urmează structura von Neumann prezentată mai jos. Aceasta este folosită pentru a defini setul de componente care există într-un sistem de calcul şi interacţiunea dintre acestea.

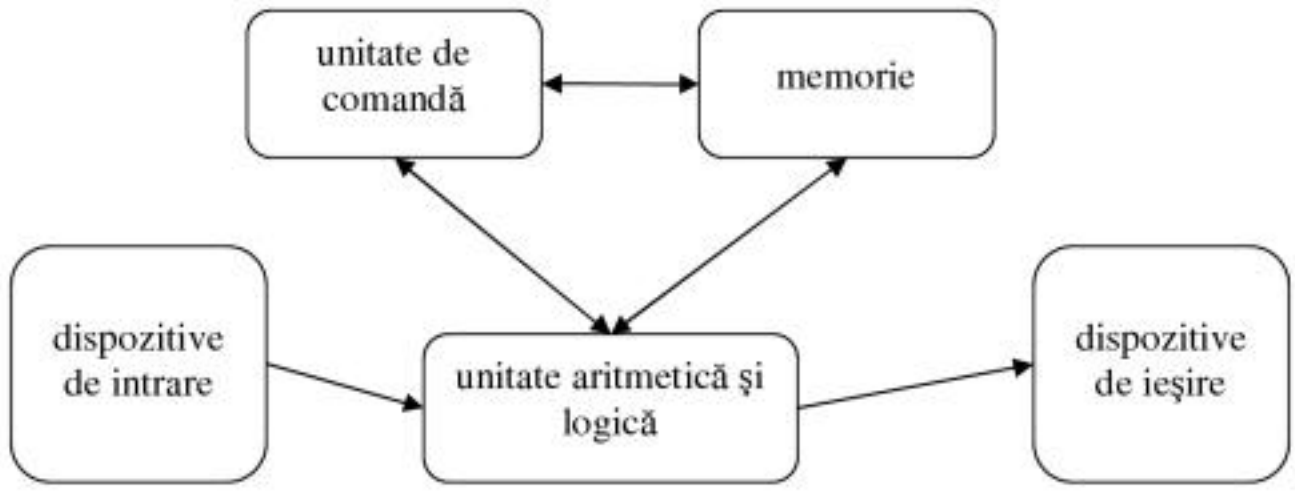


Figura 6-2 - Arhitectura von Neumann

După cum se observă, arhitectura von Neumann precizează 5 componente ale unui sistem de calcul:

- unitatea de comandă sau unitatea de control este responsabilă cu comandarea modului în care celelalte componente lucrează; unitatea de comandă va interpreta codul unui program (va decodifica instrucţiunile) şi va cere unităţii aritmetice şi logice execuţia sarcinilor necesare;
- unitatea aritmetică şi logică se mai numeşte unitatea de execuţie; rolul acesteia este de a executa codul transmis de la unitatea de comandă; unitatea aritmetică are un set de registre pentru stocarea locală a informaţiei şi un acumulator pentru efectuarea de operaţii; rolul unităţii aritmetice şi logice este îndeplinit de procesor;
- memoria este componenta în care se stochează codul programului de executat şi datele acestuia; memoria este folosită de unitatea de execuţie şi de comandă; unitatea de comandă citeşte din memorie instrucţiunile şi le decodifică, pe când unitatea de execuţie citeşte şi scrie în memorie datele cu care efectuează diverse calcule;
- dispozitivele de intrare/ieşire sunt cele care intermediază comunicaţia sistemului cu exteriorul; informaţiile de la aceste dispozitive sunt copiate în memorie şi folosite

ulterior de unitatea de comandă şi unitatea de execuţie; exemple de dispozitive periferice sunt monitorul, tastatura, mouse-ul, hard disk-ul, etc.

6.2.2 Procesorul

Procesorul sau unitatea centrală de prelucrare (CPU – Central Processing Unit) este „creierul" sistemului. Este echivalentul unităţii de comandă şi a celei de execuţie din arhitectura von Neumann. Procesorul este cel care va interpreta codul unui program şi va efectua operaţii cu datele acestuia şi este componenta de bază a execuţiei unui proces.

Procesorul este cel care determină arhitectura unui sistem de calcul. Astfel, procesoarele Intel sau AMD fac parte din arhitectura x86. Procesoarele mai noi sunt fac parte din arhitectura x86_64. Arhitectura unui sistem de calcul se referă la setul de instrucţiuni pe care procesorul le oferă utilizatorului.

Procesorul dispune de un set de locaţii de stocare de foarte mare viteză denumite registre. Un registru este folosit ca depozit local de stocare pentru acumulator. Viteza procesorului este foarte mare comparativ cu viteza memoriei şi de aceea este nevoie de registre pentru a efectua calcule rapide. În ziua de azi dimensiunea registrelor este de 32 sau de 64 de biţi. Dimensiunea registrelor este sursa denumirii sistemului. Un sistem se spune sistem pe 32 de biţi sau sistem pe 64 de biţi după dimensiunea registrelor sale.

Caracteristica vitezei unui procesor este, de obicei, frecvenţa acestuia, măsurată în Hz. În zilele noastre, frecvenţa de lucru a unui procesor este de ordinul GHz.

Exemple de procesoare sunt Intel Pentium, Intel Xeon, AMD Athlon, AMD Opteron, etc.

6.2.3 Memoria

Când ne referim la memoria unui sistem de calcul ne vom referi, de obicei, la memoria RAM (Random Access Memory). Aceasta este folosită pentru a stoca datele şi codul unui proces. În momentul în care se doreşte efectuarea unui calcul sau decodificarea unei instrucţiuni, se face un apel la memorie pentru citirea acelei informaţii.

În zilele noastre, memoria RAM folosită este de tipul DRAM (Dynamic RAM). Tipul cel mai recent este DDR3. Caracteristicile importante ale unui modul de memorie RAM sunt:

- frecvenţa de lucru: de ordinul MHz
- timpul de acces: de ordinul nanosecundelor
- rata de transfer: măsurată în GB/s
- capacitatea: actualmente de ordinul GB

6.2.3.1 Memoria cache

Din cauza diferenţei mari de viteză între memoria RAM şi procesor, se foloseşte un nivel intermediar de memorie: mai rapidă dar mai scumpă şi de dimensiunea mai mică. Această memorie poartă numele de memorie cache şi este folosită pentru reţinerea anumitor informaţii din memoria RAM.

Memoria cache foloseşte conceptul de localizare temporală şi spaţială. Aceasta înseamnă că referirea unei zone de memorie va însemna şi referirea unei zone apropiate. La fel, referirea unei zone de memorie va însemna refolosirea acesteia în momentele apropiate. În mod tipic, memoria cache are dimensiunea de ordinul sutelor de KB sau câtorva MB.

6.2.4 Placa de bază; interconectarea componentelor

Placa de bază (*mainboard, motherbord*) este componenta fizică al cărei rol este asigurarea conectivității celorlalte componente fizice. În mod tipic, o placă de bază are locuri (*sloturi, socket-uri*) pentru celelalte componente: procesor, memorie, placă video, etc.

Placa de bază conține două „nuclee" de interconectare denumite chipset-uri. Acestea sunt *northbridge* și *southbridge*. Așa cum rezultă și din denumirea lor, *northbridge*-ul se află în partea de sus a plăcii de bază iar *southbridge*-ul în partea de jos. *Northbridge*-ul este folosit pentru a conecta componentele rapide ale unui sistem de calcul: procesorul, memoria, placa video. Există o conexiune a *northbridge* cu *southbridge*-ul. *Southbridge*-ul este folosit pentru interconectarea componentelor mai lente ale sistemului: hard-disk, CD-ROM, dispozitive PCI, etc.

6.2.5 Dispozitive periferice; magistrale

Dispozitivele periferice ale sistemului asigură comunicația acestuia cu exteriorul. Conectarea acestor dispozitive periferice se realizează prin intermediul unor conectori în placa de bază. Acești conectori sunt oferiți, fie direct de placa de bază (în cazul unor plăci incorporate), fie de o placă specializată conectată pe placa de bază (placa video, placa de rețea, etc.)

Conexiunea diverselor plăci în placa de bază se realizează prin intermediul unor sloturi și a unor magistrale dedicate: PCI, AGP, USB, etc.

Magistrale importante ale sistemului sunt:

- FSB – Front Side Bus: conectează procesorul de northbridge
- Back Side Bus – conectează memoria cache de procesor
- PCI – Peripheral Component Interconnection
- AGP – Advanced Graphics Port
- PCI Express
- USB – Universal Serial Bus

6.2.6 Hard disk-ul

Probabil cel mai important dispozitiv periferic al sistemului este hard-disk-ul (HDD – hard disk drive). Acesta este folosit, de obicei, ca suport fizic pentru sistemele de fișiere. Hard-disk-urile sunt un dispozitiv de stocare permanentă (nepersistentă) a datelor.

Parametrii importanți care caracterizează un hard-disk sunt:

- capacitatea acestuia: în ziua de azi de ordinul sutelor de GB sau ordinul TB;
- viteza (măsurată în RPM – Rotations Per Minute): cu valori tipice de 5400 RPM, 7200 RPM, 10000 RPM
- dimensiunea buffer-ului (un cache folosit pentru mărirea vitezei): cu valori de ordinul MB

Există un set de interfețe și magistrale prin care un hard-disk poate fi accesat. Dintre acestea amintim ATA, Serial ATA, SCSI, SAS, etc. Fiecare dintre aceste magistrale oferă o interfață distinctă de conectare a hard-disk-ului la placa de bază și un alt protocol de comunicare cu hard-disk-ul pentru accesarea informațiilor de pe acesta.

6.3 Suportul pentru dispozitive la nivelul kernel-ului

Nucleul sistemului de operare este cel care intermediază accesul aplicațiilor la hardware. Pentru ca hardware-ul să poată fi utilizat, nucleul trebuie să știe cum să comunice cu acesta. Nucleul oferă suport de comunicație cu dispozitivele sistemului prin intermediul driver-elor.

> ***Driver**-ul este un set de rutine care se execută la nivelul nucleului și care permit accesul la un anumit dispozitiv.*

Dispozitivele pe care le poate utiliza o aplicație pot fi dispozitive fizice reale sau pot fi dispozitive virtuale (care simulează comportamentul unui dispozitiv hardware). În Linux, suportul pentru dispozitive poate fi inclus în kernel (prin compilare) sau poate fi disponibil prin intermediul *modulelor*, module care pot fi încărcate la cerere de sistem.

În prezent, majoritatea distribuțiilor conțin kernel-uri compilate cu suport generic pentru dispozitive, iar suportul pentru anumite dispozitive este disponibil ca module. Modulele sunt încărcate doar dacă dispozitivele aferente sunt prezente.

6.3.1.1 Listarea modulelor încărcate la un moment dat în sistem

Pentru a lista modulele încărcate la un moment dat se utilizează comanda `lsmod`:

```
$ lsmod
Module                  Size  Used by
rfcomm                 40856  0
l2cap                  25728  5 rfcomm
bluetooth              55908  4 rfcomm,l2cap
lp                     12452  0
cpufreq_userspace       5408  0
cpufreq_powersave       2688  0
cpufreq_ondemand        9228  0
freq_table              5792  2 cpufreq_stats,cpufreq_ondemand
[…]
```

Comanda afișează pentru fiecare modul în parte, pe coloane:

- numele;
- spațiul de memorie ocupat de acel modul;
- numărul și numele modulele care depind de el.

Același lucru se poate afla prin intermediul sistemului de fișiere virtual `/proc`. Fișierul `/proc/modules` conține lista modulelor încărcate la un moment dat.

```
$ cat /proc/modules
rfcomm 40856 0 - Live 0xe0bbb000
l2cap 25728 5 rfcomm, Live 0xe0b47000
bluetooth 55908 4 rfcomm,l2cap, Live 0xe0b51000
lp 12452 0 - Live 0xe0b39000
cpufreq_userspace 5408 0 - Live 0xe0b31000
cpufreq_stats 7360 0 - Live 0xe0b2e000
cpufreq_powersave 2688 0 - Live 0xe0859000
cpufreq_ondemand 9228 0 - Live 0xe0aee000
freq_table 5792 2 cpufreq_stats,cpufreq_ondemand, Live 0xe0b27000
[…]
```

Modulele disponibile în sistem se găsesc în `/lib/modules/<versiune_kernel>`. Modulele incluse în nucleul distribuției se găsesc în `/lib/modules/<versiune_kernel>/kernel`. În acest director ele sunt organizate în funcție de categoria din care fac parte – există astfel directoarele `arch` (module pentru arhitectura curenta), `crypto` (criptografie), `drivers` (driver-e dispozitive), `fs` (sisteme de fișiere), `lib` (module care implementează diferite funcții în kernel), `net` (rețea), `security` (securitate), `sound` (sunet):

```
# ls /lib/modules/2.6.18-4-686/kernel/
arch  crypto  drivers  fs  lib  net  sound
```

Versiunea de kernel poate fi aflată utilizând comanda:

```
$ uname -r
2.6.20-15-generic
```

După cum s-a observat şi din ieşirea comenzii `lsmod`, între module există dependinţe. Aceste dependinţe sunt descrise în `/lib/modules/<versiune_kernel>/modules.dep` – acest fişier este generat automat la rularea comenzii *depmod*.

6.3.1.2 Încărcarea unui modul

Înainte de a prezenta modul în care un modul este încărcat în nucleu, trebuie menţionat că în distribuţiile moderne Linux modulele se încarcă automat în funcţie de dispozitivele hardware prezente în sistem.

Încărcarea manuală a unui modul se poate realiza folosind comanda *insmod* sau comanda `modprobe`. Comanda `insmod` primeste ca argument calea către fişierul unde se găseşte modulul. Acest lucru o face destul de greu de folosit. Un exemplu de utilizare este dat mai jos (se presupune că modulul ce se doreşte a fi încărcat nu este deja încărcat).

```
# insmod ./drivers/input/misc/pcspkr.ko
# lsmod | grep pcsp
pcspkr                  4224  0
```

O comandă echivalentă comenzii `insmod` utilizate mai sus dar care încarcă şi dependenţele unui modul (dacă există) este `modprobe <nume_modul>`:

```
# modprobe pcspkr
```

Pentru a configura `modprobe` se poate utiliza fişierul `/etc/modprobe.conf` sau directorul `/etc/modprobe.d`.

6.3.1.3 Descărcarea unui modul din kernel

Pentru descărcarea unui modul din kernel se poate utiliza comanda `rmmod` sau comanda `modprobe -r`. Comenzile următoare descarcă modulul `pcspkr`, cu observaţia că, la utilizarea lui `modprobe`, se încearcă şi descărcarea modulelor de care `pcspkr` depinde (dacă există şi nu sunt folosite de alte module).

```
# rmmod pcspkr
# modprobe -r pcspkr
```

6.4 Analiza hardware a unui sistem (magistrale, chipset, CPU, memorie, dispozitive)

6.4.1 Lista hardware - /sys

Folosind sistemul de fişiere virtual (*sysfs*) montat de kernel în `/sys`, se pot obţine informaţii despre dispozitivele hardware aflate în system. `/sys` este populat de kernel cu informaţii provenite de la driver-ele încărcate la un moment dat. Dispozitivele hardware afişate în sys sunt cele care au fost detectate de kernel şi care sunt utilizabile de acesta. Tabelul următor prezintă conţinutul directoarelor din `/sys`.

Director	Conținut
/sys/devices	Subdirectoarele care reflectă modul fizic de conectare al dispozitivelor în sistem
/sys/bus	Legături simbolice grupate către dispozitive grupate în funcție de bus-ul de care aparțin dispozitivele
/sys/class	Dispozitivele grupate în funcție de clasa din care fac parte
/sys/block	Dispozitivele bloc
/sys/firmware	Dispozitive cu suport firmware
/sys/fs	Informații despre sistemele de fișiere
/sys/kernel	Informații despre kernel
/sys/module	Subdirectoare pentru fiecare modul în parte
/sys/power	Informații despre consumul de curent

Figura 6-3 Ierarhia din /sys

6.4.2 Comenzi pentru afişarea dispozitivelor şi a stării acestora

Informații despre dispozitivele din sistem şi starea acestora pot fi aflate şi prin intermediul unor comenzi. În cele ce urmează, se vor prezenta comenzile utilizate pentru a prezenta diverse informații despre dispozitivele din sistem.

6.4.2.1 Dispozitive USB

Pentru a afişa magistralele şi dispozitivele USB prezente în sistem se utilizează comanda `lsusb`:

```
$ lsusb
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
```

Informații despre magistrala USB se regăsesc şi în `/sys/bus/usb/`. Subdirectorul `devices` prezintă dispozitivele sub formă de legături către directoare din `/sys/devices`. Subdirectorul `drivers` din `/sys/bus/usb/` conține informații despre driver-ele utilizate.

```
$ ls -l /sys/bus/usb/devices/
total 0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 1-0:1.0 ->
../../../devices/pci0000:00/0000:00:07.2/usb1/1-0:1.0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 2-0:1.0 ->
../../../devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb2/2-0:1.0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 usb1 ->
../../../devices/pci0000:00/0000:00:07.2/usb1
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 usb2 ->
../../../devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb2
$ ls -l /sys/bus/usb/drivers/
total 0
drwxr-xr-x 2 root root 0 2007-09-04 18:16 hub
drwxr-xr-x 2 root root 0 2007-09-04 18:16 usb
drwxr-xr-x 2 root root 0 2007-09-04 18:16 usbfs
$ ls -lR /sys/bus/usb/drivers/ | grep module
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
```

Ultima comandă listează recursiv directoarele din `/sys/bus/usb/drivers`, căutând liniile care conțin referințe la fişiere cu numele `module`. Se observă astfel că modulul utilizat este `usbcore` (datele şi orele fişierelor au fost eliminate din ieşirea comenzii).

6.4.2.2 Dispozitive PCI

Dispozitivele de pe magistrala PCI se pot afla utilizând comanda `lspci`:

```
$ lspci
00:00.0 Host bridge: Intel Corporation 82865G/PE/P DRAM Controller/Host-
Hub Interface (rev 02)
00:01.0 PCI bridge: Intel Corporation 82865G/PE/P PCI to AGP Controller
(rev 02)
00:1d.0 USB Controller: Intel Corporation 82801EB/ER (ICH5/ICH5R) USB UHCI
Controller #1 (rev 02)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev c2)
00:1f.0 ISA bridge: Intel Corporation 82801EB/ER (ICH5/ICH5R) LPC
Interface Bridge (rev 02)
 [...]
```

Similar magistralei USB, se pot determina şi directoarele din `/sys` asociate dispozitivelor PCI. Comanda următoare prezintă modul în care se poe afla modulele utilizate de dispozitivele PCI, folosind aceeaşi procedură ca în cazul dispozitivelor USB (datele şi orele fişierelor au fost eliminate din ieşirea comenzii).

```
$ ls -lR /sys/bus/pci/drivers/ | grep module
lrwxrwxrwx 1 root root    0 module -> ../../../../module/snd_intel8x0
lrwxrwxrwx 1 root root    0 module -> ../../../../module/intel_agp
lrwxrwxrwx 1 root root    0 module -> ../../../../module/ata_generic
lrwxrwxrwx 1 root root    0 module -> ../../../../module/ata_piix
 [...]
```

6.4.2.3 Senzori

O altă sursă de informaţii despre hardware-ul prezent într-un sistem o reprezintă senzorii. Pachetul `lm-sensors` include programul `sensors` care afişează informaţii citite de la aceştia.

După instalarea pachetului trebuie rulată comanda `sensors-detect` care va detecta senzorii prezenţi în sistem si va indica ce module trebuie încărcate. După ce configurarea se realizează cu succes, poate fi utilizată comanda `sensors`.

```
$ sensors
smsc47m192-i2c-0-2d
Adapter: SMBus I801 adapter at 1400
+2.5V:      +2.60 V  (min =  +0.00 V, max =  +3.32 V)
VCore:      +1.42 V  (min =  +0.00 V, max =  +2.99 V)
+3.3V:      +3.13 V  (min =  +0.00 V, max =  +4.38 V)
+5V:        +5.16 V  (min =  +0.00 V, max =  +6.64 V)
[...]
smsc47m1-isa-0800
Adapter: ISA adapter
fan1:      2792 RPM  (min =  640 RPM, div = 8)
fan2:      2118 RPM  (min =  640 RPM, div = 8)
```

6.4.2.4 Sistemul ACPI

Sistemele moderne au implementat sistemul ACPI[16] (*Advanced Configuration and Power Interface*) pentru configurarea unor dispozitive şi managementul consumului de curent. Fişierele virtuale ale acestui sistem se află în `/proc/acpi`.

Spre exemplu, sistemul ACPI oferă printre altele informaţii despre:

- procsoarele din sistem – `/proc/acpi/processor`
- ventilatoarele din sistem – `/proc/acpi/fan`
- bateriile din sistem – `/proc/acpi/battery`
- senzorii de temperatura – `/proc/acpi/thermal_zone`

[16] *ACPI – http://www.acpi.info*

6.4.3 smartmontools

Pachetul *smartmontools* pune la dispoziție programe pentru monitorizarea hard-disk-urilor utilizând tehnologia S.M.A.R.T. (*Self-Monitoring, Analysis and Reporing Technology*). În unele sisteme, trebuie activat suportul S.M.A.R.T. în BIOS (unele BIOS-uri au aceasta configurare în setul de opțiuni avansate). Tehnologia permite monitorizarea indicatorilor de stare a hard-disk-urilor în scopul anticipării posibilelor căderi.

În pachet se găsesc două programe:

- `smartctl` - pentru analiza stării S.M.A.R.T. la un moment dat
- `smartd` - un daemon pentru monitorizarea permanentă.

Pentru a verifica starea la un moment dat pentru primul hard-disk, se poate utiliza comanda:

```
  # smartctl --all /dev/sda
  smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce Allen
  Home page is http://smartmontools.sourceforge.net/

  === START OF INFORMATION SECTION ===
  Model Family:     Seagate Barracuda 7200.7 and 7200.7 Plus family
  Device Model:     ST3120827AS
  Serial Number:    4MS03F43
  Firmware Version: 3.42
  User Capacity:    120,033,041,920 bytes
  […]

  SMART Attributes Data Structure revision number: 10
  Vendor Specific SMART Attributes with Thresholds:
  ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED
WHEN_FAILED RAW_VALUE
    1 Raw_Read_Error_Rate     0x000f   062   049   006    Pre-fail  Always
-       16
    3 Spin_Up_Time            0x0003   098   096   000    Pre-fail  Always
-       0
    4 Start_Stop_Count        0x0032   100   100   020    Old_age   Always
-       659
  […]
```

SMART are şi suport pentru execuția de teste asupra discului. Există teste lungi şi teste scurte. Execuția unui test poate fi urmărită în timp real dar poate fi şi programată. Rezultatele testului se pot afla în ieşirea comenzii `smartctl -all /dev/sda`.

Un test scurt poate fi programat utilizând comanda:

```
  # smartctl --test=short /dev/sda
  smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce Allen
  Home page is http://smartmontools.sourceforge.net/

  === START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
  Sending command: "Execute SMART Short self-test routine immediately in
off-line mode".
  Drive command "Execute SMART Short self-test routine immediately in off-
line mode" successful.
  Testing has begun.
  Please wait 1 minutes for test to complete.
  Test will complete after Thu Sep 13 10:34:15 2007

  Use smartctl -X to abort test.
```

Se poate observa la o nouă rulare a comenzii `smartctl -all /dev/sda` că apar la ieşire si rezultatele testului (dacă s-a terminat), precum în exemplul de mai jos.

```
# smartctl --all /dev/sda
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF INFORMATION SECTION ===
[…]

=== START OF READ SMART DATA SECTION ===
SMART overall-health self-assessment test result: PASSED

[…]
```

Pe lânga testul scurt mai există şi alte teste (spre exemplu un test lung şi un test offline). Mai multe exemple de folosire a lui `smartctl` se pot afla din `man smartctl` şi din `smartctl -h`.

Daemon-ul `smartd` monitorizează hard-disk-urile şi permite înregistrarea stării acestora şi a defectelor apărute prin intermediul `syslog`. În mod implicit verifică discurile la fiecare 30 minute. Daemon-ul se poate configura prin intermediul fişierului `/etc/smard.conf`. Pentru lucrul cu daemoni se poate consulta capitolul [vezi 4.3.4]

6.5 Interfaţa cu dispozivitele din userspace - udev şi /dev

În Linux totul este un fişier. Există câte un fişier pentru fiecare dispozitiv în directorul `/dev`.

Fişierele din `/dev` sunt create de managerul de dispozitive numit *udev*. Acesta este executat în întregime ca daemon în spaţiul utilizator (*userspace*) şi astfel poate executa programe arbitrare la apariţia sau la dispariţia dispozitivelor din sistem. *udev* suportă nume persistente pentru dispozitive, ceea ce înseamnă că poate fi configurat ca pentru acelaşi dispozitiv să folosească întotdeauna acelaşi nume pentru fişierul asociat dispozitivului (în implementările mai vechi pentru `/dev`, la reintroducerea în sistem a unor dispozitive, acestea puteau primi nume diferite comparativ cu introducerea anterioară).

udev primeşte notificări de la kernel atunci când apar sau dispar dispozitive. Astfel, spre exemplu, la conectarea unui USB stick în calculator, kernel-ul notifică *udev* că a apărut un nou dispozitiv de stocare. În funcţie de schema de nume utilizată, *udev* poate, spre exemplu, să aleagă ca nume de dispozitiv următorul nume disponibil după numele hard-disk-ul prezent în sistem (presupunând ca dispozitivul asociat hard-disk-ului este `/dev/sda`, *udev* alege ca nume `/dev/sdb`). După ce este ales un nume de dispozitiv, se poate executa un program care, în cazul prezentat aici, să monteze sistemul de fişiere de pe USB stick într-un subdirector din `/media`. Tot în funcţie de configurări, anumite dispozitive pot să aibă fişierele asociate într-un subdirector din `/dev`.

Pe lângă execuţia de programe, *udev* mai permite modificarea permisiunilor pentru fişierele din `/dev` create. Asfel se poate configura ca pentru orice scanner introdus în sistem, fişierul creat sa aparţină grupului cu numele scanner. În acest fel, numai utilizatorii care fac parte din acest grup vor putea lucra cu acest fişier şi implicit cu scanner-ul. Trebuie amintit faptul că un utilizator poate să fie membru în mai multe grupuri.

Configurările pentru *udev* se găsesc în `/etc/udev/rules.d`. Acest director conţine fişiere care descriu modul în care, în funcţie de dispozitiv, se formează numele dispozitivelor, se dau permisiuni fişierelor sau se execută programe. De obicei acest director conţine configurări aplicabile în orice situaţie şi nu se recomandă editarea fişierelor din el.

6.6 Lucrul cu dispozitive (/dev):

S-a menționat anterior că udev se ocupă și de stabilirea permisiunilor pentru anumite dispozitive. Un utilizator trebuie să se găsească în grupul din care face fișierul dispozitivului pentru a putea utiliza dispozitivul (prin intermediul fișierului, bineînțeles).

Tabelul de mai jos prezintă câteva nume asociate în mod normal cu dispozitivele într-un mediu Linux. Trebuie menționat că de obicei numele sunt urmate de un număr sau o literă, care indică al câtelea dispozitiv de acel tip este în sistem. Spre exemplu, două unități de dischetă vor fi numite `/dev/fd0` și `/dev/fd1`, iar două hard-disk-uri vor fi numite `/dev/sda` și `/dev/sdb`.

Nume	Dispozitiv
`fd`	Floppy-disk
`hd`	Hard-disk IDE
`lp`	Imprimantă
`sd`	Hard-disk SATA (SCSI)
`tty`	Terminal fizic
`ram`	Dispozitiv care reprezintă memoria RAM
`loop`	Dispozitiv utilizat pentru montarea fișierelor precum dispozitivele reale

Figura 6-4 Convenția de nume pentru dispozitive în Linux

Comanda următoare prezintă conținutul tipic pentru `/dev`:

```
$ ls -l /dev/
total 0
lrwxrwxrwx 1 root root 10 2007-08-31 14:48 adsp -> sound/adsp
lrwxrwxrwx 1 root root 12 2007-08-31 14:48 agpgart -> misc/agpgart
lrwxrwxrwx 1 root root 11 2007-08-31 14:48 audio -> sound/audio
drwxr-xr-x 3 root root  0 2007-08-31 14:48 bus
drwxr-xr-x 2 root root  0 2007-08-31 14:48 cd
lrwxrwxrwx 1 root root 17 2007-08-31 14:48 cdrom -> /dev/cd/cdrom-sr0
lrwxrwxrwx 1 root root 17 2007-08-31 14:48 cdrom0-> /dev/cd/cdrom-sr0
[...]
drwxr-xr-x 2 root root  0 2007-08-31 14:48 loop
lrwxrwxrwx 1 root root  6 2007-08-31 14:48 loop0 -> loop/0
lrwxrwxrwx 1 root root  6 2007-08-31 14:48 loop1 -> loop/1
crw-r----- 1 root kmem  1,    1 2007-08-31 14:48 mem
lrwxrwxrwx 1 root root 11 2007-08-31 14:48 mixer -> sound/mixer
crw-rw-rw- 1 root root  1,    3 2007-08-31 14:48 null
[...]
crw-rw-rw- 1 root root  1,    8 2007-08-31 14:48 random
brw-rw---- 1 root disk  8,    0 2007-08-31 14:48 sda
brw-rw---- 1 root disk  8,    1 2007-08-31 14:48 sda1
brw-rw---- 1 root disk  8,    2 2007-08-31 14:48 sda2
brw-rw---- 1 root disk  8,   16 2007-08-31 14:48 sdb
 [...]
```

6.6.1 Tipuri de dispozitive

În Linux, dispozitive se împart în 2 categorii, fiecare categorie cu particularitățile ei:

- dispozitive caracter
- dispozitive bloc

Dispozitivele pot fi diferențiate după litera care apare pe prima coloană în ieșirea comenzii `ls -l`, coloană care indică tipul fișierului. În lista de mai sus fișierele care au tipul „`c`” indică dispozitive de tip caracter iar fișierele cu tipul „`b`” indică dispozitive de tip bloc.

Câteva dispozitive sunt generate de kernel şi au un comportament aparte. Aceste dispozitive sunt numite pseudo-dispozitive.

6.6.1.1 Dispozitivele caracter

Dispozitivele caracter *corespund dispozitivelor care transmit date câte un caracter o dată.*

Aceste dispozitive sunt de obicei folosite pentru a transmite fluxuri de date, precum:

- terminalul - `/dev/tty0, /dev/tty1 ...`
- porturile seriale - `/dev/ttyS0, /dev/ttyS1 ...`
- dispozitivele audio - `/dev/sequencer, /dev/mixer, /dev/midi`

6.6.1.2 Dispozitivele bloc

Dispozitivele bloc *corespund dispozitivelor care lucrează cu datele la nivel de bloc (scriu, citesc, şterg blocuri de date).*

Dispozitivele care funcţionează în acest fel sunt dispozitivele de stocare:

- fizice (hard disk-uri, unităţi CD-ROM/DVD-ROM etc.) - `/dev/sda, /dev/sda1, /dev/hda ...`
- virtuale (dispozitivele loop) - `/dev/loop0`
- asociate zonelor de memorie - `/dev/ram0 ...`

6.6.1.3 Pseudo-dispozitive

Există o serie de dispozitive prezente în `/dev` dar care nu corespund unor componente hardware. Ele sunt echivalente unor dispozitive virtuale dar implementează comportamentul dispozitivelor normale de tip caracter.

Cele mai cunoscute sunt:

- `/dev/null` – fişier în care se poate scrie orice şi totul este pierdut – de obicei ieşirile standard sau de eroare ale programelor sunt redirectate către `/dev/null` pentru a suprima afişarea oricărui mesaj;
- `/dev/zero` – fişier din care se poate citi la infinit octetul cu valoarea 0 – de obicei este folosit pentru a suprascrie zone de memorie/disc cu zerouri sau pentru a umple fişiere;
- `/dev/urandom` – fişier din care se pot citi la infinit octeţi cu valori aleatoare – utilizat de obicei pentru a scrie fişiere sau zone de pe disc cu date aleatoare.

Câteva exemple de utilizare sunt prezentate mai jos.

Suprimarea ieşirii standard pentru un program:

```
$ ./program > /dev/null
```

Suprimarea ieşirii de eroare pentru un program (mesajele de eroare se pierd):

```
$ ./program 2>/dev/null
```

Suprimarea ambelor ieşiri (standard şi eroare) pentru un program:

```
$ ./program 2>&1 >/dev/null
```

Suprascrierea primei partiţii de pe primul hard-disk cu zerouri:

```
# dd if=/dev/zero of=/dev/sda1
```

Suprascrierea primului hard-disk cu date aleatoare:

```
# dd if=/dev/urandom of=/dev/sda
```

6.6.2 Manipularea datelor la nivel scăzut (comanda dd)

Pentru a lucra cu datele la nivel de octet sistemele de operare de tip Unix pun la dispoziție comanda `dd`. Această comandă poate fi utilizată pentru a manipula date de orice dimensiune. Se bazează pe faptul că datele sunt transferate de la o sursă la o destinație, eventual cu simple transformări.

Comanda primeşte sursa şi destinația sub forma a doi parametri:

- **sursa**: `if=<fişier_intrare>` (daca lipseşte acest parametru se presupune că sursa este intrarea standard)
- **destinația**: `of=<fişier_ieşire>` (daca lipseşte acest parametru se presupune că destinația este ieşirea standard)

Fişierele pot fi atât fişiere normale cât si fişiere care reprezintă dispozitive bloc sau dispozitive caracter. Astfel, o utilizare foarte simplă a comenzii `dd` este următoarea:

```
$ dd if=/dev/zero of="myfile"
```

Comanda de mai sus copiază de la intrare la ieşire octeți până când apare o eroare, din moment ce nu a fost specificată nici o limita. În aceste condiții, comanda se termină doar dacă fişierul sursă e limitat sau fişierul destinație e limitat. În cazul de mai sus, sursa are dimensiune infinită iar destinația e limitată doar de dimensiunea spațiului liber disponibil pe partiția unde se scrie fişierul.

Pentru a specifica dimensiunea care se doreşte a fi copiata se utilizează o combinație a următorilor parametri:

- **numărul de blocuri**: `count=<număr_blocuri>` - dacă acest parametru lipseşte copierea se opreşte doar în condițiile prezentate mai sus
- **dimensiunea unui bloc**: `bs=<dimensiune_în_octeți>` (*block size*) – pot fi folosiți şi multiplii, spre exemplu: `bs=1048576, bs=1024K, bs=1M` sunt toate echivalente; dacă lipseşte acest parametru se utilizează dimensiunea implicită a unui bloc de 512 octeți (dimensiunile blocurilor citite şi dimensiunile blocurilor scrise pot fi ajustate individual prin utilizarea parametrilor `ibs` si `obs`, ele sunt modificate simultan prin utilizarea parametrului `bs`)

Exemplele următoare ilustrează utilizarea comenzii `dd` împreună cu parametrii precizați anterior:

- copierea de 10 MB de date din 10 blocuri de 1 MB (1024K); copierea se face din `/dev/zero` către `/dev/null` deci nu are nici un efect:

```
$ dd if=/dev/zero of=/dev/null bs=1024K count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.00490124 seconds, 2.1 GB/s
```

- acelaşi efect ca mai sus doar că se precizează în M în loc de K:

```
$ dd if=/dev/zero of=/dev/null bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.00509437 seconds, 2.1 GB/s
```

- copierea de 10MB de date în blocuri de 1M la intrare şi 128K la ieşire:

```
$ dd if=/dev/zero of=/dev/null count=10 ibs=1M obs=128K
10+0 records in
80+0 records out
10485760 bytes (10 MB) copied, 0.00622419 seconds, 1.7 GB/s
```

- Comanda `dd` permite şi poziționarea în fişierele utilizate pentru intrare şi ieşire:
- intrare: `skip=<număr_blocuri>` - sare `<număr_blocuri>` din fişierul de intrare, echivalentul a `<număr_blocuri>*ibs` octeți
- ieşire: `seek=<număr_blocuri>` - sare `<număr_blocuri>` din fişierul de intrare, echivalentul a `<număr_blocuri>*obs` octeți

Poziționarea este utilă în cazul în care se doreşte ca de pe dispozitivul de intrare să se citească o anumită zonă, precum un anumit sector de pe hard disk. Exemplul următor ilustrează acest lucru – se sare peste primul sector şi se citesc două sectoare de pe hard disk (trebuie menționat că dimensiunea unui sector pe HDD este 512 octeți, valoarea implicită pentru parametrul `bs`):

```
# dd if=/dev/sda of=/dev/null skip=1 count=2
2+0 records in
2+0 records out
1024 bytes (1.0 kB) copied, 0.043731 seconds, 23.4 kB/s
```

6.7 Analiza proceselor si a kernel-ului

6.7.1 Informații despre procese

Kernel-ul Linux pune la dispoziție un mod foarte simplu de a afla informații despre procesele care rulează la un moment dat în sistem. Folosind sistemul de fişiere *proc* montat în `/proc`, kernel-ul oferă o interfață la informațiile despre procese.

În directorul `/proc` se găsesc directoare cu numele alcătuit numai din cifre care identifică fiecare câte un proces. Numele unui director reprezintă de fapt PID-ul unui proces. Se formează astfel o structură ierarhică de forma `/proc/PID/tip_informaţie` unde *tip_informaţie* este un fişier care conține date despre proces. Aceste fişiere se regăsesc pentru fiecare proces în parte. Câteva exemple de fişiere din `/proc/PID` şi conținutul lor sunt prezentate în tabelul de mai jos.

Fişier	Conținut
`cwd`	Legătură către directorul de unde a fost pornit procesul (*current working directory*)
`cmdline`	Linia de comandă utilizată la pornirea procesului
`environ`	Mediul de lucru (valoarea variabilelor de mediu) de la pornirea programului
`exe`	Legătură către fişierul executabil al procesului
`fds`	Director care conține legături cu nume numeric – fiecare legătură are numele unui descriptor de fişier şi indică fişierul (sau dispozitivul pentru majoritatea comenzilor din consola) asociat descriptorului.
`mounts`	Lista dispozitivelor montate la momentul execuției procesului
`root`	Legătură către rădăcina sistemului de fişiere văzut de proces – un proces poate să pornească şi cu considerând o altă rădăcină pentru sistemul de fişiere
`status`	Statistici despre proces, precum memoria ocupată

Figura 6-5 Fişierele din ierarhia /proc/PID

6.7.2 Informații despre sistem

Prin intermediul directorului `/proc`, kernel-ul expune și o serie de informații legate de sistem. Tabelul următor prezintă fișierele mai importante din `/proc` și conținutul acestora.

Fișier	Conținut
`/proc/cmdline`	Linia care este folosita la inițializarea kernel-ului, inclusiv parametrii trimiși acestuia
`/proc/cpuinfo`	Informații despre procesorul/procesoarele din sistem
`/proc/devices`	Lista dispozitivelor caracter și a celor bloc
`/proc/interrupts`	Statistici despre numărul de întreruperi și dispozitivele asociate întreruperilor
`/proc/meminfo`	Informații despre utilizarea memoriei
`/proc/modules`	Lista modulelor încărcate la un moment dat în sistem – comanda *`lsmod`* prezintă conținutul fișierului într-un mod user-friendly
`/proc/partitions`	Lista partițiilor
`/proc/swaps`	Lista dispozitivelor pe care se face swap
`/proc/uptime`	Timpul de la ultima pornire, în secunde – comanda *`uptime`* prezintă conținutul fișierului într-un mod user-friendly
`/proc/version`	Versiunea kernel-ului care rulează la un moment dat

Figura 6-6 Informații expuse prin intermediul lui /proc

Conținutul pentru oricare din fișierele prezentate mai sus se poate afla folosind comanda *`cat`*, precum în exemplul de mai jos:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Intel(R) Core(TM)2 Duo CPU     T7500  @ 2.20GHz
stepping        : 8
cpu MHz         : 2194.893
cache size      : 4096 KB
fdiv_bug        : no
[…]
```

Oricare dintre aceste fișiere poate fi folosit pentru ca, la un moment dat, să se găsească o informație despre sistem. Spre exemplu, dacă se dorește aflarea tipului procesorului/procesoarelor din sistem se poate utiliza comanda următoare:

```
$ cat /proc/cpuinfo | grep "model name"
model name      : Intel(R) Core(TM)2 Duo CPU     T7500  @ 2.20GHz
```

În `/proc` se găsește un director special cu numele *self* care conține informații despre procesul care acceseaza la momentul respectic directorul `/proc`. De asemenea, fișierul *mounts* din `/proc` este un link către fișierul *mounts* al procesului care acceseaza `/proc`.

```
$ ls -l /proc
total 0
[…]
dr-xr-xr-x  5 ubuntu  ubunt   0 2007-09-06 01:41 13373
lrwxrwxrwx  1 root    root   11 2007-09-06 01:41 mounts -> self/mounts
lrwxrwxrwx  1 root    root   64 2007-09-03 17:41 self -> 13373
```

Prin utilizarea directorului self din `/proc`, un proces poate afla informații despre starea proprie, consultând fișierele din directorul *self* (echivalent directorului ce are drept nume PID-ul procesului).

Este important de reținut că `/proc` expune informații care sunt accesibile și prin intermediul unor comenzi. Aceste informații, fie că sunt citite direct din `/proc` fie că sunt citite prin intermediul comenzilor, pot fi filtrate pentru a se extrage date utile pentru utilizator.

6.7.3 Kernel "tunables"

Sysctl este o interfață care permite configurarea unor parametri din kernel în timpul execuției acestuia. Parametrii pot fi modificați în două feluri:

- prin modificarea directa a fișierelor din `/proc/sys`
- prin utilizarea comenzii sysctl

Primul caz presupune scrierea si citirea parametrilor din anumite căi, precum în exemplul următor:

```
# cat /proc/sys/net/ipv4/ip_forward
0
# echo "1" > /proc/sys/net/ipv4/ip_forward
# cat /proc/sys/net/ipv4/ip_forward
1
```

Comenzile de mai sus au activat rutarea pachetelor de rețea la nivelul kernel-ului, scriind „1” în `/proc/sys/net/ipv4/ip_forward`. Această facilitate este utilă dacă se dorește ca un sistem care are acces Internet să permită accesul unei întregi subrețele.

Folosind comanda `sysctl`, același lucru se obține în felul următor:

```
# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
# sysctl -w net.ipv4.ip_forward="1"
net.ipv4.ip_forward = 1
```

Se observă că pentru a afișa starea unui parametru se folosește o adresare asemănătoare cu cea a structurilor din C/C++. Comanda a doua modifică un parametru prin utilizarea opțiunii `-w` (*write*).

Modificările prezentate anterior sunt modificări temporare, ele fiind pierdute în cazul în care se repornește sistemul. Pentru a efectua configurări permanente, acestea se scriu in fișierul `/etc/sysctl.conf`. Presupunând ca se dorește dezactivarea forwarding-ului, sintaxa în fișier este destul de simplă (liniile care încep cu # din fișier sunt comentarii):

```
$ cat /etc/sysctl.conf
[…]
# Controls IP packet forwarding
net.ipv4.ip_forward = 1
[…]
```

Fișierul de configurare este citit la fiecare pornire a sistemului. Pentru a forța citirea fișierului de configurare, se poate utiliza comanda `sysctl -p`:

```
# sysctl -p
net.ipv4.ip_forward = 1
```

6.8 Studii de caz:

6.8.1 Backup și restaurare pentru MBR și tabela extinsă de partiții

MBR-ul (*Master Boot Record*) reține informații despre partițiile primare aflate pe un hard-disk precum atribute ale acestora, poziția lor pe disc si dimensiunea. MBR-ul mai conține și bootloader-ul – un mic program care pornește un sistem de operare prezent pe computer. Într-

un mediu mixt Windos/Linux, la o instalare a Windows-ului se va observa că Linux-ul devine inaccesibil (bootloader-ul Windows-ului nu cunoaşte nimic despre Linux). În aceste condiţii, pentru a preîntâmpina acest lucru, se recomandă realizarea unui back-up pentru MBR. Mai multe detalii despre MBR găsiţi în capitolul 5.

Backup-ul MBR-ului de pe primul HDD se realizează cu următoarea comandă:

```
# dd if=/dev/sda of=mbr.img bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.8004e-05 seconds, 10.7 MB/s
```

În urma execuţiei comenzii se realizează fişierul `mbr.img` de 512 octeţi cu MBR-ul. Acest fişier poate fi plasat într-un loc sigur şi accesibil în cazul în care discul devine inaccesibil (pe un USB stick sau CD-ROM spre exemplu).

Pentru a restaura MBR-ul, se efectuează operaţia inversă:

```
# dd if=mbr.img of=/dev/null bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.622e-05 seconds, 11.1 MB/s
```

În MBR se poate stabili ca una dintre partiţii să fie partiţie extinsă. În partiţia extinsă pot fi create mai multe partiţii logice, pentru o mai bună organizare a discului. Informaţiile despre partiţiile logice nu sunt ţinute în MBR, deoarece nu încap în cei 512 octeţi. Partiţiile logice sunt reţinute sub formă de lista înlănţuita în afara MBR-ului.

În momentul în care se lucrează cu schema de partiţii de pe un hard disk, se recomanda realizarea unui backup înainte. Modificarea schemei de partiţii nu şterge datele de pe hard disk, doar schimbă descrierea partiţiilor (locul unde se găsesc acestea). În situaţia în care se şterge o partiţie importantă (fie ea logică sau primară), este util să existe un backup al acestor informaţii pentru recuperare. Modul de backup pentru MBR a fost descris anterior.

Pentru a realiza backup-ul configuraţiei tuturor partiţiilor se utilizează programul sfdisk. Acesta analizează atât la partiţiile primare cât şi la cele logice, dar nu se uită şi la informaţiile care nu sunt legate de partiţionare din MBR (precum bootloader). El este un bun complement pe lângă metoda amintită mai sus de a face backup la MBR.

Comanda următoare face backup pentru întreaga schemă de partiţionare utilizând sfdisk – fişierul de backup este /usbstick/partitions.backup:

```
# sfdisk -d /dev/sda > /usbstick/partitions.backup
```

Pentru a restaura schema de partiţionare salvată anterior, sfdisk se utilizează astfel:

```
# sfdisk /dev/sda < /usbstick/partitions.bacup
```

6.8.2 Backup al unui disc cu sectoare inaccesibile

Există situaţii când, de obicei după o perioada mai lungă de utilizare, un hard disk începe să nu mai funcţioneze corect. Se observă acest lucru din erorile pe care sistemul de operare le afişează când vrea să citească date de pe disc. În asemenea situaţii, cea mai rapidă metodă de backup este copierea fişierelor pe un alt disc.

Acesta este cazul favorabil, când datele mai pot fi identificate şi sistemul de operare mai funcţionează. În cazul în care defectul hard disk-ului apare în zona în care este descris sistemul de fişiere, exista o probabilitate destul de mare ca datele să nu mai poată fi salvate prin simpla copiere a fişierelor. În aceasta situaţie se poate opta pentru repararea sistemului de fişiere utilizând fsck, dar nu se doreşte ca aceasta reparare să aibă loc pe hard disk-ul defect. Întâi se realizează o copie a tuturor datelor de pe hard disk pe alt hard disk, copie numită imagine.

Comanda utilizată pentru a realiza această imagine este `dd`, împreună cu parametrul `conv=noerror`, parametru care îi spune dd-ului să nu se oprească atunci când detectează un bloc inaccesibil.

```
# dd if=/dev/sda of=/dev/sdb conv=noerror
```

Execuția acestei comenzi poate dura foarte mult timp, datorită numărului mare de blocuri care trebuie citite. Pentru a accelera execuția comenzii, se poate mări dimensiunea blocului citit/scris prin intermediul parametrului `bs` (`block size`), prezentat anterior. Utilizarea unui bloc mai mare decât dimensiunea unui sector de hard disc (512 octeți) poate duce la pierderea ultimelor sectoare de pe discul sursă, căci nu se poate realiza un bloc cu ele.

6.8.3 Crearea unei imagini de CD; montarea unei imagini

O imagine de CD se creează în mod asemănător creării unei imagini de hard disk.

```
# dd if=/dev/hda of=cd.iso
```

Această comandă va citi CD-ul introdus în unitatea `/dev/hda` sector cu sector şi va scrie imaginea lui în fişierul `cd.iso`.

Pentru a putea utiliza imaginea, aceasta trebuie montată într-un director precum orice alt sistem de fişiere, dar cu următoarele observații:

- tipul sistemului de fişiere de pe un CD este *iso9660;*
- deoarece fişierul nu este un dispozitiv real, trebuie utilizată şi opțiunea `loop`.

```
# mount -t iso9660 -o loop cd.iso /mnt/cd
```

Pentru a demonta imaginea, se utilizează comanda:

```
# umount /mnt/cd
```

În acest fel pot fi păstrate pe disc imagini ale CD-urilor utilizate des şi/sau pot fi testate imagini de CD create de diferite utilitare.

Trebuie menționat că se poate aplica aceeaşi procedură şi pentru DVD-uri, atâta timp cât sistemul de fişiere este corect precizat în comanda mount (DVD-urile pot fi scrise cu formatul UDF)

6.8.4 Utilizarea unui fisier de pe o partiție FAT32 ca fişier de swap pentru un LiveCD Linux

Se observă de multe ori că LiveCD-urile funcționează greu pe sisteme care nu au multă memorie RAM disponibilă. Într-un sistem instalat, memoria RAM poate fi ajutată şi de prezența spațiului de swap de pe hard disk. Într-un sistem pe care nu există nici o partiție de swap, un LiveCD nu activează nici un astfel de spațiu. O posibilitate o reprezintă utilizarea unui fişier de swap, pe o partiție deja existentă în sistem. S-a utilizat pentru exemplificare o partiție FAT32, dar se poate utiliza orice tip de partiție pe care Linux-ul poate scrie.

Pentru început, s-a montat sistemul de fişiere (partiția a doua, primul hard disk) unde se doreşte crearea fişierului de swap:

```
# mount -t vfat /dev/sda2 /mnt/temp
```

În acest moment se poate crea un fişier de swap de dimensiunea dorită utilizând programul `dd`:

```
# dd if=/dev/zero of=/mnt/temp/fisier.swap bs=1M count=256
256+0 records in
256+0 records out
268435456 bytes (268 MB) copied, 6.41171 seconds, 41.9 MB/s
```

S-a creat un fişier de 256 MB (plin cu zerouri) pentru a fi folosit ca fişier de swap. În acest moment trebuie transformat fişierul în zonă de swap, într-un proces asemănător formatării partiţiilor:

```
# mkswap /mnt/temp/fisier.swap
```

În acest moment, fişierul este pregătit pentru a fi folosit ca zonă de swap. Sistemul de operare trebuie să afle de prezenţa fişierului de swap printr-o operaţie asemănătoare montării unui sistem de fişiere:

```
# swapon /mnt/temp/fisier.swap
```

După execuţia acestei comenzi sistemul foloseşte fişierul creat mai sus ca zonă de swap.

Cuvinte cheie

- kernel
- kernel-space
- user-space
- maşină virtuală
- von Neumann
- CPU, procesor
- RAM, DRAM
- placă de bază, chipset, northbridge, southbridge
- hard-disk, HDD
- magistrală, PCI, AGP
- dispozitiv
- modul
- driver
- lsmod, insmod, modprobe, rmmod
- /sys
- lsusb, lspci
- sensors
- smartmontools
- /dev
- udev
- dispozitive de tip bloc
- dispozitive de tip caracter
- pseudodispozitive
- dd
- /proc
- sysctl
- /etc/sysctl.conf
- MBR

Întrebări

Care din următoarele nu este o componentă a arhitecturii von Neumann?

- ❑ CPU
- ❑ unitatea de comandă
- ❑ memoria
- ❑ dispozitivele de intrare ieşire

Care utilitar poate fi folosit pentru efectuarea unui backup la MBR?

- ❑ lspci
- ❑ lsusb
- ❑ sysctl
- ❑ dd

Care din următoarele nu oferă direct informații despre sistem?

- /sys
- /proc
- dd
- lsusb

Care comandă nu este folosită pentru interacțiunea cu modulele din kernel?

- lsmod
- modprobe
- modinstall
- insmod

Care din următoarele nu este un exemplu de pseudodispozitiv?

- /dev/null
- /dev/zero
- /dev/hda3
- /dev/urandom

Care din următoarele dispozitive nu este asociat unui hard-disk?

- /dev/sda1
- /dev/hda5
- /dev/sda
- /dev/ttyS0

Care din următoarele nu este un tip de dispozitiv?

- dispozitiv fizic
- pseudodispozitiv
- dispozitiv de tip bloc
- dispozitiv de procesor

Care din următoarele acronime nu este corelat direct cu dispozitive fizice?

- SCSI
- USB
- ACPI
- TGZ

7 Configurări de rețea de bază

You know it's love when you memorize her IP address to skip DNS overhead.

Ce se învață din acest capitol?

- Noțiunea de rețea de calculatoare
- Topologia unei rețele de calculatoare
- Adresarea IP
- Parametrii unei rețele de calculatoare: adresă IP, mască de rețea, gateway
- Configurarea unei interfețe de rețea în Linux; adăugarea de rute în Linux
- Configurare temporară; configurare permanentă; configurare statică; configurare dinamică
- Configurare DNS
- Verificarea conectivității într-o rețea locală
- Configurarea rețelei în Windows
- Configurarea PPPoE în Linux

7.1 Concepte de rețea

Apariția şi evoluția calculatoarelor au dus la dorința şi nevoia naturală de a le pune să comunice între ele pentru a partaja date mai uşor sau pentru a putea îndeplini împreună aceeaşi sarcină. Deşi dezvoltarea rețelelor de calculatoare a fost inițial lentă, implementarea lor a cunoscut o creştere mare după 1990, o dată cu explozia numărului de calculatoare legate la Internet.

Dacă inițial rețelele de calculatoare au fost create pentru a facilita schimbul de informație în mediul academic, în momentul de față ele se adresează în mare măsură utilizatorilor obişnuiți şi mediului comercial.

Pentru ca un calculator să fie conectat la o rețea trebuie realizate două conexiuni: o conexiune fizică si o conexiune logică.

Conexiunea fizică este utilizată pentru a codifica informația sub forma de semnale electrice (în cazul cel mai uzual întâlnit conexiunea se realizează prin cablu UTP), semnale optice (pentru legăturile prin fibra optică) sau semnale electromagnetice (pentru legăturile fără fir). În primele două cazuri ea se realizează printr-un cablu care leagă calculatorul de un echipament de rețea.

Conexiunea logică cuprinde un set de mecanisme de adresare complexe (ce asigură comunicația între calculatoarele conectate la rețea), cuprinde controlul fluxului de date şi oferă garanția integrității datelor. Cel mai cunoscut mecanism de adresare prezent în cadrul conexiunii logice este IP (Internet Protocol). Acesta asigură identificarea unică şi comunicarea calculatoarelor atât în rețelele locale, cât şi în Internet.

Rețelele de calculatoare, din punctul de vedere al standardelor folosite, sunt împărțite în trei mari categorii: rețele locale de calculatoare (*Local Area Network* - LAN), rețele metropolitane de calculatoare (*Metropolitan Area Network* - MAN) şi rețele de calculatoare pe arii extinse (*Wide Area Network* - WAN). În momentul de față această clasificare nu se mai păstrează 100%, rețelele MAN dispărând aproape complet şi fiind înlocuite cu rețele LAN.

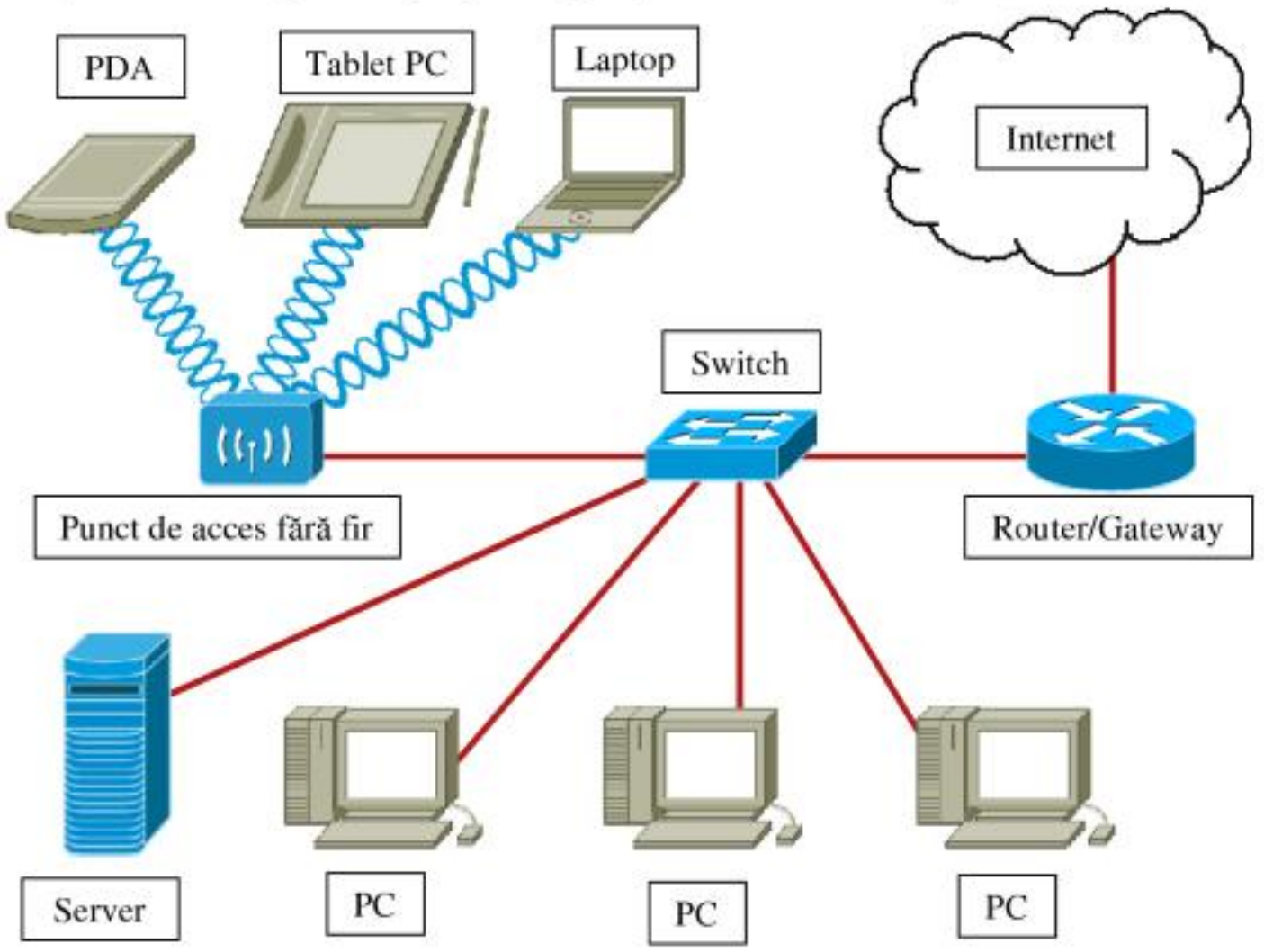


Figura 7-1 Schema unei rețele locale de calculatoare

O rețea locală de calculatoare este formată cel mai adesea din mai multe stații legate între ele. O stație poate fi un calculator, un laptop, un PDA, un *Tablet PC*, un server, un ruter, etc.

Conectarea stațiilor se realizează printr-un echipament de rețea numit **switch**. Acesta transferă informația între calculatoare bazându-se pe adresa MAC.

***Adresa MAC** este un număr cu o lungime de 48 de biți care identifică în mod unic fiecare interfață de rețea Ethernet.*

Prin urmare nu pot exista două interfețe de rețea Ethernet cu aceeaşi adresa MAC. Adresa MAC este înscrisă din fabrică într-o memorie ROM de pe placă şi ea nu poate fi schimbată.

Datorită faptului că protocolul DHCP de configurare automată a adreselor IP utilizează frecvent adresa MAC pentru a aloca o adresă IP unui calculator, este uzuală practica de a modifica adresa MAC a unei plăci de rețea. Acest lucru nu înseamnă rescrierea memoriei ROM ci specificarea unei alte adrese pe care sistemul de operare să o folosească în locul celei inițiale.

Switch-ul permite comunicația între calculatoare aflate în aceeaşi rețea locală. Pentru a putea accesa şi alte calculatoare (cel mai adesea pentru a putea accesa Internetul) trebuie ca unul din echipamentele din rețea să aibă o a doua legătură la o altă rețea. Acest echipament poate sa fie o stație sau un server cu mai multe interfețe de rețea sau un echipament de rețea dedicat numit **router**. Transferul informației din rețeaua locală mai departe înspre Internet se bazează pe adresa IP.

***Adresa IP** este un număr cu o lungime de 32 de biți.*

O stație poate avea mai multe adrese IP, în mod uzual câte o adresă pentru fiecare interfață. Este posibil însă ca pe o interfață să fie configurate mai multe adrese IP.

Moduri de adresare: adresare unicast, multicast, broadcast

Comunicațiile între echipamente pot fi împărțite în trei categorii funcție de numărul de destinatari:

- comunicații **unicast**: au o sursă şi un destinatar
- comunicații **multicast**: au o sursă şi mai mulți destinatari ce fac parte dintr-un grup specific
- comunicații **broadcast**: au o sursă şi ca destinatar toate stațiile dintr-o rețea

Atât mesajele de tip multicast cât şi cele de tip broadcast sunt adresate mai multor destinatari simultan. Diferența dintre ele constă în faptul că mesajele de tip broadcast se adresează tuturor stațiilor dintr-o rețea, pe când cele multicast se adresează doar unora din aceste stații.

7.2 Parametri de rețea

7.2.1 Adresă IP şi mască de rețea

Adresa IP pe 32 de biți a fost introdusă în perioada '80, când nu se preconiza creşterea spectaculoasă a Internetului. După 1990, datorită dezvoltării rețelelor, cerințele pentru adrese IP au crescut, astfel încât nu peste mult timp adresele au început să se epuizeze. Din acest motiv au apărut mai multe mecanisme menite să reducă numărul de adrese IP utilizate, printre care adresele IP private şi translatarea de adrese IP (*Network Address Translation* - NAT). În paralel a fost dezvoltată şi o soluție pe termen lung: protocolul IPv6 a cărui utilizare va creşte în anii următori. În cadrul acestui protocol adresa IP are 128 de biți.

Pentru a uşura utilizarea adreselor IP, cei 32 de biți sunt separați în patru grupuri a câte 8 biți, fiecare grup fiind scris în formă zecimală. Astfel adresa IP utilizată în mod uzual este compusă din patru numere zecimale între 0 şi 255 despărțite prin puncte:

```
01011001110101110001111010000111 - cei 32 de biți ai unei adrese IP
01011001.11010111.00011110.10000111 - patru grupuri a câte 8 biți
89.215.30.135 - fiecare grup este transformat în format zecimal
```

Atunci când o interfață de rețea are configurată o adresă IP, adresa are două componente: o parte din cei 32 de biți reprezintă adresa rețelei din care face parte stația şi cealaltă parte a biților reprezintă adresa stației în cadrul rețelei. Sistemul este similar cu cel utilizat pentru codul poştal: în cadrul codului 014288, primele două cifre pot reprezenta județul, următoarele două pot reprezenta oraşul, iar ultimele două pot să reprezenta strada. Astfel destinația este localizată în arii din ce în ce mai restrânse.

Pentru a putea spune care dintre cei 32 de biți reprezintă adresa rețelei şi care reprezintă adresa calculatorului în cadrul rețelei, este utilizată o **mască de rețea**. Aceasta este tot un număr de 32 de biți cu o caracteristică specială: este compusă, pornind de la stânga la dreapta, dintr-un bloc compact de 1 urmat de un bloc compact de 0. Pentru utilizarea mai uşoară a măştii de rețea este folosită aceeaşi notație ca şi în cadrul adresei IP:

```
11111111000000000000000000000000 - cei 32 de biți ai unei măşti
11111111.00000000.00000000.00000000 - patru grupuri a câte 8 biți
255.0.0.0 - fiecare grup este tranformat în format zecimal
```

În cadrul măştii de reţea un bit egal cu 1 reprezintă faptul că bitul corespunzător din adresa IP face parte din adresa reţelei, iar un bit egal cu 0 reprezintă faptul că bitul corespunzător din adresa IP face parte din adresa calculatorului în cadrul reţelei:

```
01011001.11010111.00011110.10000111 - adresă IP
11111111.00000000.00000000.00000000 - mască de reţea
<-A.R.-> <-------+A.S.------------>
A.R.- Adresa Reţelei
A.S. Adresa Staţiei
```

Pentru a putea afla din ce reţea face parte o adresă IP, se face ŞI logic pe biţi între adresa IP şi masca de reţea. După realizarea operaţiei de ŞI logic se obţine o adresa IP care are toţi biţii din partea de adresă a reţelei egali cu adresa reţelei din care face parte staţia, şi toţi biţii din partea de adresa a staţiei în cadrul reţelei egali cu 0:

```
01011001.11010111.00011110.10000111 = 89.215.30.135 - adresă IP
11111111.00000000.00000000.00000000 = 255.0.0.0 - mască de reţea
----------------------------------- - ŞI logic pe biţi
01011001.00000000.00000000.00000000 = 89.0.0.0 - adresa reţelei
<-A.R.-> <-------A. C.------------>
01011001.11111111.11111111.11111111 = 89.255.255.255 - adr. broadcast
```

Adresa IP în care toţi biţii din partea de adresă a staţiei sunt egali cu 0 se numeşte **adresa reţelei** din care face parte staţia. Toate staţiile dintr-o reţea locală au adrese IP care au aceeaşi adresă a reţelei. În felul acesta se realizează corespondenţa între conexiunea fizică la reţea şi cea logică.

Adresa IP în care toţi biţii din partea de adresă a staţiei în cadrul reţelei sunt egali cu 1 se numeşte **adresa de broadcast** a reţelei. Dacă un pachet este trimis către această adresă, atunci el va fi procesat de toate calculatoarele din reţea.

Clase de adrese IP

Conceptul de clase de adrese IP nu este unul fundamental pentru utilizarea sistemelor de operare. În schimb în cadrul reţelelor de calculatoare el are un rol important pentru modul în care funcţionează o reţea.

Spaţiul de adrese IP este împărţit în cinci categorii, denumite clase. Apartenenţa unei adrese IP la una din clase se stabileşte după valoarea primului octet (ca atare după valoarea primului număr zecimal din notaţia cu punct).

- În cazul în care primul octet are o valoare cuprinsă între 0 şi 127, adresa IP face parte din clasa A.
- În cazul în care primul octet are o valoare cuprinsă între 128 şi 191, adresa IP face parte din clasa B.
- În cazul în care primul octet are o valoare cuprinsă între 192 şi 223, adresa IP face parte din clasa C.
- În cazul în care primul octet are o valoare cuprinsă între 224 şi 240, adresa IP face parte din clasa D
- În cazul în care primul octet are o valoare cuprinsă între 224 şi 240, adresa IP face parte din clasa E

```
Clasa A: 0.0.0.0 - 127.255.255.255
Clasa B: 128.0.0.0 - 191.255.255.255
Clasa C: 192.0.0.0 - 223.255.255.255
Clasa D: 224.0.0.0 - 239.255.255.255
Clasa E: 240.0.0.0 - 255.255.255.255
```

Adresele publice de IP (configurate pe interfeţele staţiilor şi folosite în adresarea unicast) fac parte din clasele A, B şi C. Adresele IP din clasa D sunt folosite pentru multicast, iar cele din clasa E sunt rezervate.

Fiecare din clasele de adrese A, B şi C au definită o mască standard de rețea care se foloseşte împreună cu adresele IP din clasa respectivă: IP-urile din clasa A au asociată o mască ce conține 8 biți egali cu 1, cele din clasa B au asociată o mască ce conține 16 biți egali cu 1 şi cele din clasa C au asociata o mască ce conține 24 de biți egali cu 1.

```
255.0.0.0 - masca de rețea specifică clasei A
255.255.0.0 - masca de rețea specifică clasei B
255.255.255.0 - masca de rețea specifică clasei C
```

În momentul de față de foarte puține ori se mai foloseşte masca de rețea standard pentru o clasa anume. Din motive care țin de eficiența modului de utilizare a adreselor IP, măştile de rețea nu mai ai doar trei lungimi fixe, ci pot avea o lungime variabilă.

Adrese IP publice şi adrese IP private

În momentul în care numărul de adrese IP rămase libere a început să scadă simțitor la începutul anilor '90, au apărut mecanisme menite să rezolve parțial această problema. Printre mecanismele apărute se numără şi spațiul de adrese private.

Din fiecare clasă de adrese a fost rezervat un spațiu de adrese care nu pot fi accesate direct din afara rețelei locale. Practic stațiile care au configurate adrese IP private sunt invizibile din afara rețelei lor, şi implicit din Internet. De aceea pot exista mai multe stații, făcând parte din rețele diferite, care au aceeaşi adresa IP privată. În felul acesta se realizează o economie mare de adrese IP.

Spațiile de adrese private sunt:

```
10.0.0.0 - 10.255.255.255 din clasa A
170.16.0.0 - 172.31.255.255 din clasa B
192.168.0.0 - 192.168.255.255 din clasa C
```

Pentru ca o stație ce are configurată o adresă IP privată să poată accesa stații care nu se află în aceeaşi rețea cu ea este necesară utilizarea mecanismelor de NAT (*Network Address Translation*). De cele mai multe ori, prin NAT, toate stațiile dintr-o rețea cu adrese IP private vor fi recunoscute în afara rețelei ca având aceeaşi adresă IP publică.

7.2.2 Ruter implicit (default gateway)

Pentru a putea accesa calculatoare aflate în alte rețele, este necesar ca unul dintre echipamentele conectate în rețea să aibă o a doua interfață conectată la o altă rețea. Din acea a doua rețea se poate realiza mai departe accesul către Internet. Acest echipament se numeşte *gateway*.

Atunci când o stație A doreşte să comunice cu o stație B, primul pas pe care îl face este să verifice dacă adresa IP a lui B este în aceeaşi rețea cu adresa lui A. În cazul în care cele două stații sunt în aceeaşi rețea, informațiile sunt trimise direct către B. Dacă B nu se află în aceeaşi rețea cu A, atunci informațiile sunt trimise către gateway, urmând ca acesta să găsească o rută către B.

Prin urmare este necesar ca fiecare stație din rețea să cunoască adresa gateway-ului pentru a putea avea acces în afara rețelei şi masca de rețea pentru a vedea dacă este cazul să folosească gateway-ul sau nu.

7.2.3 DNS

Un utilizator accesează în mod normal un număr mare de servicii de rețea: cele mai multe oferite de diverse servere. Accesarea fiecărui serviciu este condiționată de cunoaştere adresei IP a serverului care oferă serviciul respectiv. Spre exemplu, pentru a accesa o pagină web, este necesară cunoaşterea adresei IP a serverului pe care este găzduită acea pagină.

Pentru că memorarea unui număr mare de adrese IP nu este un lucru comod şi uşor de ealizat, a apărut serviciul numit Domain Name System (DNS). În cadrul acestui serviciu se realizează corespondența între un şir de caractere şi o adresă IP. De aceea DNS-ul poate fi privit ca o carte de telefon de unde, dacă se cunoaşte numele unui server, i se poate afla adresa IP.

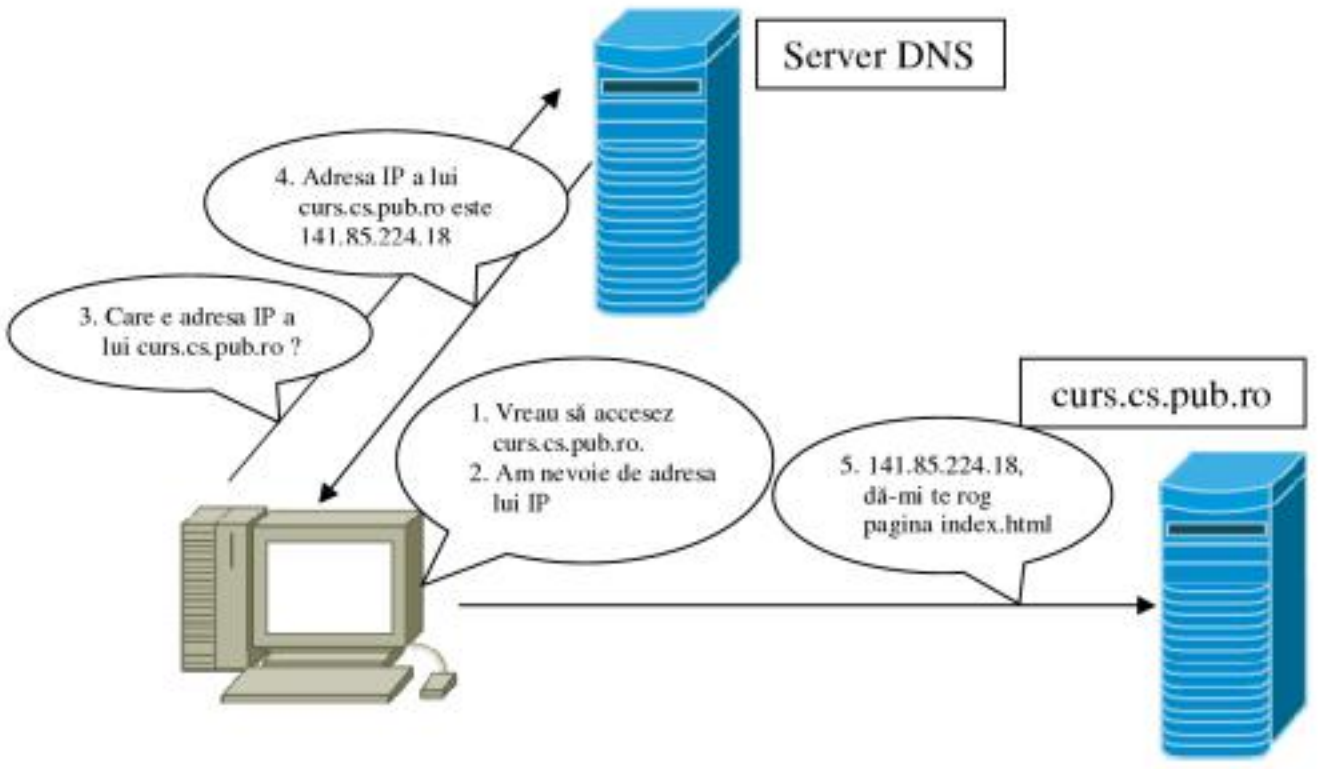


Figura 7-2 Rolul unui server DNS

Un nume de domeniu (asociat unui calculator) este compus din mai multe părți despărțite prin puncte: de exemplu curs.cs.pub.ro. Cea din extremitatea dreaptă se numeşte *Top Level Domain* şi poate conține indicativul unei țări (.ro pentru România, .us pentru SUA, .jp pentru Japonia, etc) sau poate reprezenta un anumit tip de organizație (de exemplu .com pentru organizațiile comerciale, .org pentru organizațiile non-profit, .mil pentru armata SUA, etc).

În stânga Top Level Domain se găsesc subdomenii. În exemplul de mai sus subdomeniul pub este asociat Universității Politehnica Bucureşti, cs este asociat Facultății de Automatică şi Calculatoare, Catedra de Calculatoare.

Serviciile de DNS sunt oferite de servere specializate. Pentru a putea accesa un astfel de server, este necesară cunoaşterea adresei lui IP. Exista servere publice care oferă servicii de DNS, însă de cele mai multe ori este de preferat ca în cadrul rețelei locale să existe un server de DNS din motive ce țin de timpul de răspuns.

Chiar dacă serverul de DNS este plasat în rețeaua locală, DNS-ul rămâne cel mai lent serviciu de rețea şi componenta cu cel mai mare timp de răspuns dintre toate componentele ce alcătuiesc mecanismele de comunicație în rețea. De aceea se spune că „*You know it's love when you memorize her IP address to skip DNS overhead*".

Este posibil uneori ca adresa IP asociată unei stații să fie dinamică. Aceasta înseamnă că din timp în timp stația respectivă va primi o altă adresă IP. Pentru a putea accesa stația fără să i se cunoască adresa IP existentă la un moment dat se foloseşte sistemul numit DynDNS (*Dynamic DNS*). În cadrul acestui sistem stația va trimite către un server DNS specializat informații actualizate legate de adresa ei IP.

7.3 Configurări temporare

7.3.1 Interfețe de rețea. Configurări permanente şi configurări temporare

Din punct de vedere logic, fiecărei placi de rețea îi corespunde în cadrul sistemului de operare o interfață. Pe această interfață se configurează o adresă IP şi o mască de rețea.

În Linux interfețele de rețea conțin în denumirea lor două componente. Prima parte reprezintă tipul interfeței (de exemplu `eth` pentru Ethernel, `wlan` pentru placile de rețea fără fir, etc) si numărul ei (`eth0` reprezintă prima interfață de Ethernet, `wlan1` reprezintă a două interfață fără fir, etc).

Configurarea unei adrese IP se poate realiza în două moduri: configurare temporară şi configurare permanentă. Fiecare din cele două se poate realiza la rândul ei static (parametrii sunt introduşi manual de administrator) sau dinamic (parametrii se configurează în mod automat).

Configurarea temporară statică se realizează cu ajutorul comenzii `ifconfig` (a cărei denumire vine de la *interface configuration*). Efectul aceste configurări este imediat (parametrii se aplica imediat după ce comanda este dată) însă o dată cu repornirea sistemului de operare, configurările se pierd.

Configurarea temporară dinamică se realizează cu ajutorul unui client DHCP, de exemplu `dhclient` sau `dhcpcd`.

Configurarea permanentă statică sau dinamică se realizează cu ajutorul fişierului `/etc/network/interfaces`. Efectul acestei comenzi nu este imediat (configurările nu se aplică o dată cu salvarea fişierului), însă configurările se păstrează şi după repornirea sistemului de operare. Acest lucru se petrece deoarece fişierul în care sunt trecute configurările este citit şi interpretat de sistemul de operare la inițializare, configurările din fişier fiind din nou aplicate.

	Configurare statică	Configurare dinamică
Configurare temporară	ifconfig	Client DHCP: dhclient, dhcpd, etc
Configurare permanentă	/etc/network/interfaces	/etc/network/interfaces

Figura 7-3 Metode de configurare a interfețelor de rețea

Şi configurarea unui gateway se poate realiza în mod temporar cu ajutorul comenzii `route` sau în mod permanent prin fişierul de configurare `/etc/network/interfaces`.

Fiecare din cele două metode (permanent vs. temporar) are avantajele sale. Configurările temporare au efect imediat si se aplică uşor. Cele permanente se pot aplica automat la fiecare inițializare a sistemului de operare.

7.3.2 Configurarea temporară statică a unei adrese IP pe o interfață

Pentru a configura o interfață de rețea în mod temporar se foloseşte comanda `ifconfig`. Modul de utilizare al comenzii este următorul:

```
# ifconfig [-v] [-a] [-s] [interface]
# ifconfig [-v] interface [aftype] options | address ...
```

Prin utilizarea comenzii fără nici un parametru se pot afla informații despre interfețele de rețea active:

```
asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::214:d1ff:fe38:73a6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18982017 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27864548 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:144959484 (130 MiB)  TX bytes:21665362 (20.6 MiB)
          Interrupt:18 Base address:0x4400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:5777 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5777 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:360919 (352.4 KiB)  TX bytes:360919 (352.4 KiB)
```

Pe lângă interfețele corespunzătoare plăcilor de rețea există și interfață `lo`. Numele acestei interfețe vine de la *loopback*. Orice informație se transmite pe loopback, se va întoarce înapoi tot pe loopback.

Interfața de loopback este o interfață virtuală (fără un corespondent fizic) care este folosită în scopuri de testare. Ea are asociată o adresă IP standard: 127.0.0.1. Dacă un calculator nu are nici o placa de rețea, interfața de loopback va fi singura existentă.

Parametrii uzuali ai comenzii `ifconfig` sunt următorii:

- `-v` afișează informații detaliate în cazul erorilor
- `-a` afișează informații despre toate interfețele existente fie că sunt active sau inactive

```
asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

asgard:/home/george# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
eth1      Link encap:Ethernet  HWaddr 00:18:F3:AB:45:62
          BROADCAST MULTICAST  MTU:1500  Metric:1
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

- `-s` afișează statistici legate de interfețele de rețea in format compact, sub forma unui tabel:

```
asgard:/home/george# ifconfig -s
iface   MTU Met  RX-OK RX-ERR   [...]  TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500 0   193488      0  [...] 282347      0      0      0 BMRU
lo    16436 0     5849      0  [...]    584      0      0      0 LRU
```

- `interface` reprezintă numele interfeței despre care se doresc informații sau numele interfeței care va fi configurată. De exemplu, pentru a afișa informații despre interfața de rețea `eth0` se folosește comanda

```
asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

- `address` reprezintă adresa IP care va fi configurată pe interfață. De exemplu, pentru a configura adresa IP `10.1.1.3` pe interfața `eth0`:

```
asgard:/home/george# ifconfig eth0 10.1.1.3
asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.255.255.255  Mask:255.0.0.0
          [...]
```

- `options` reprezintă opțiuni avansate ce pot fi configurate

Opțiunile cele mai des folosite sunt următoarele

- `down` dezactivează o interfață. Odată dezactivată, interfața nu va mai trimite şi nu va mai primi niciun pachet

```
asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

asgard:/home/george# ifconfig eth0 down
asgard:/home/george# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

se poate observa faptul că, după ce interfața a fost dezactivată, ea nu mai apare în lista de interfețe active. Pentru a afişa toate interfețele existente se foloseşte parametrul `-a`.

- `up` este utilizată pentru a activa o interfață. În cazul în care este specificată o adresa IP pentru a fi configurată pe interfață, opțiunea `up` este considerată implicit, chiar dacă nu este specificată

```
asgard:/home/george# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

asgard:/home/george# ifconfig eth0 up
asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

- `netmask addr` specifică masca de rețea asociata interfeței. În cazul în care această opțiune nu este prezentă se va considera masca implicită clasei IP-uri din care face parte adresa IP a interfeței

Se poate observa din exemplele anterioare faptul că, nespecificând masca de rețea, aceasta a fost considerată implicit `255.0.0.0`, care este valoarea pentru clasa A de adrese IP din care face parte `10.1.1.3`. În cazul în care se specifică şi valoarea măştii ca opțiune, rezultatul va fi:

```
asgard:/home/george# ifconfig eth0 10.1.1.3 netmask 255.255.255.0
asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

- `hw class addr` specifică utilizarea unei alte adrese MAC şi nu cea din memoria ROM a plăcii de rețea, în cazul în care driverul plăcii de rețea suportă acest lucru. Pentru a putea specifica o altă adresă MAC interfața de rețea trebuie să fie inactivă:

```
asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]

asgard:/home/george# ifconfig eth0 down
asgard:/home/george# ifconfig eth0 hw ether 01:02:03:04:05:06 up
asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 01:02:03:04:05:06
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

Utilitatea folosirii unei adrese MAC diferită de cea a plăcii de rețea apare datorită faptului că protocolul DHCP de configurare automată a adreselor IP utilizează frecvent adresa MAC pentru a aloca o adresă IP unui calculator. Pentru a obține o anumită adresă IP, trebuie folosită adresa MAC asociată în cadrul serverului DHCP cu adresa IP dorită.

7.3.3 Asigurare conectivității la Internet. Configurarea temporară statică a unei rute implicite

Comanda `ifconfig` permite doar configurarea unei adrese IP pe o interfață. Pentru a putea accesa Internetul este nevoie să fie specificată adresa unui gateway.

Specificarea temporară a unui gateway se realizează cu ajutorul unei rute. O **rută** reprezintă calea pe care o urmează pachetele în drumul lor către destinație. Rutele sunt reținute în cadrul unui structuri numite **tabelă de rutare**. Aceasta conține practic informații despre toate destinațiile accesibile la un moment dat.

Printre informațiile reținute pentru fiecare rută se numără: adresa destinației (poate fi adresa unei stații sau adresa unei întregi rețele), masca de rețea asociată şi adresa echipamentului de rețea către care se trimit informațiile pentru a ajunge la destinație.

Configurarea tabelei de rutare se realizează cu ajutorul comenzii `route`. Aceasta permite adăugarea, modificarea şi ştergerea de rute. Pentru a vedea care sunt rutele existente la un moment dat (ca atare pentru a vedea conținutul tabelei de rutare) se utilizează comanda `route` fără parametri:

```
asgard:/home/george# route
Kernel IP routing table
Destination   Gateway      Genmask         Flags Metric Ref  Use Iface
localnet      *            255.255.255.0   U     0      0      0 eth0
```

În rezultatul de mai sus se vede faptul că singura rută existentă este cea asociată rețelei locale (`localnet`) .Această rută (care este practic asociată cu rețeaua locală direct conectată) este introdusă în tabela de rutare după ce interfața de rețea asociată rețelei este configurată.

Formatul comenzii utilizat pentru adăugarea unei rute implicite este următorul:

```
# route add default gateway addr
```

Configurarea unei rute implicite se realizează astfel:

```
asgard:/home/george# route add default gateway 10.1.1.1
asgard:/home/george# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
localnet        *               255.255.255.0   U     0      0        0 eth0
default         10.1.1.1        0.0.0.0         UG    0      0        0 eth0
```

Unul dintre parametrii utili ai comenzii `route` este `-n` (*numeric*) Folosind acest parametru adresele din tabela de rutare vor fi afişat în format numeric în loc să se afişeze numele serverelor sau staţiilor. Este folosit atunci când se doreşte interogarea unui server DNS (de exemplu atunci când nu este configurat un astfel de server).

```
asgard:/home/george# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.1.0        *               255.255.255.0   U     0      0        0 eth0
0.0.0.0         10.1.1.1        0.0.0.0         UG    0      0        0 eth0
```

7.4 Configurări permanente

7.4.1 Configurarea permanentă statică a unei interfețe de rețea

Configurările de rețea permanente se realizează în cadrul fişierului `/etc/network/interfaces`. Manualul de utilizare al acestui fişier se poate accesa cu comanda:

```
man interfaces
```

Un exemplu de fişier de configurare este următorul:

```
asgard:/home/george# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo eth0
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
        address 10.1.1.2
        netmask 255.255.255.0
        network 10.1.1.0
        broadcast 10.1.1.255
        gateway 10.1.1.1
        dns-nameservers 10.1.1.1
```

Fişierul `/etc/network/interfaces` conţine diverse configurări ale interfeţelor de reţea, configurări care sunt interpretate de către utilitarele `ifup` şi `ifdown`. `ifup` este un utilitar ce permite activarea unei interfeţe de reţea. El este similar opţiunii `up` a comenzii `ifconfig`. `ifdown` dezactivează o interfaţă de reţea, fiind similar cu opţiunea `down` a comenzii `ifconfig`.

În cadrul fişierului `interfaces` se găsesc mai multe declaraţii de tipul `iface`, `auto` sau `allow`.

Declaraţiile `auto` specifică ce interfeţe sunt activate atunci când `ifup` este folosit împreună cu parametrul `-a`. Un astfel de apel (`ifup -a`) este folosit la pornirea sistemului de operare. Pornirea unei interfeţe marcate ca `auto` este realizată în momentul în care `udev` [ref udev] a creat mecanismele necesare funcţionării interfeţei.

Declarațiile `allow` specifică interfețele care sunt activate automat de către diverse subsisteme. Astfel activarea nu mai este realizată după ce `udev` a detectat placa de rețea, ci în momentul în care aceasta devine disponibilă.

Pentru specificarea unei interfețe de loopback se folosește următoarea linie:

```
iface lo inet loopback
```

După declarația `iface` urmează numele interfeței, apoi tipul de adrese pe care interfața îl folosește (`inet` reprezintă adresare IP, `inet6` reprezintă adresare IPv6). Pentru a marca o interfață ca loopback, se folosește parametrul `loopback`.

O linie asemănătoare este folosită și pentru a specifică orice alta interfață existentă. Spre exemplu, pentru configurarea interfeței `eth0`:

```
iface eth0 inet static
```

După declarația `iface` urmează numele interfeței apoi tipul de adrese pe care îl folosește interfața, iar la final, prin includerea parametrului `static` se specifică faptul că interfața este configurată manual de către administrator. O altă metodă de configurare este cea în care se folosește DHCP.

Dacă parametrul `static` a fost precizat în declarația `iface`, atunci după această declarație trebuie să urmeze parametrii cu care va fi configurată interfața de rețea:

```
iface eth0 inet static
        address 10.1.1.2
        netmask 255.255.255.0
        network 10.1.1.0
        broadcast 10.1.1.255
        gateway 10.1.1.1
        dns-nameservers 10.1.1.1
```

`address` specifică adresa IP a interfeței (parametru obligatoriu), `netmask` specifică masca de rețea care va fi folosită (parametru obligatoriu), `network` specifica adresa rețelei din care face parte interfața (parametru obligatoriu doar pentru kernelul 2.0.x [ref kernel]), adresă ce se poate obține făcând SI logic pe biți între adresa IP și masca de rețea, `broadcast` specifică adresa de broadcast a rețelei din care face parte interfața (parametru opțional).

7.4.2 Configurarea permanentă statică a unei rute implicite

O a doua modalitate de specificare a rutei implicite este utilizarea fișierului `/etc/network/interfaces`. Structura acestuia a fost descrisă în paragraful anterior.

Printre parametrii care se pot configura după declarația `iface` se află și gateway-ul implicit. Specificarea acestuia se realizează astfel:

```
iface eth0 inet static
        [...]
        gateway 10.1.1.1
```

unde `10.1.1.1` este adresa IP a gateway-ului.

7.4.3 DHCP. Configurarea unei interfețe în mod automat

Configurarea parametrilor unei interfețe de rețea se poate realiza static sau dinamic.

În cazul configurărilor statice toți parametrii trebuie introduși manual pe fiecare stație din rețea. În cazul în care se schimbă gateway-ul de exemplu, pe fiecare stație din rețea trebuie realizată această schimbare.

Configurările dinamice permit atribuirea automată a parametrilor de rețea, printre care adresa IP, masca de rețea, gateway-ul, serverul de DNS.

Cel mai utilizat protocol de configurare dinamică (automată) este **DHCP** (*Dynamic Host Configuration Protocol*). Pentru ca acest protocol să funcționeze este necesar ca în rețeaua locală să se găsească un server de DHCP. Serverului îi sunt specificate plaja de adrese IP pe care le poate atribui calculatoarelor şi restul configurărilor de rețea.

Atunci când o stație dorește să obțină o adresă IP, va trimite un mesaj către toată rețeaua (broadcast) prin care încearcă să determine dacă există un server de DHCP. Dacă acest server există, atunci el va trimite configurațiile către stație.

Configurarea parametrilor interfeței de rețea folosind DHCP se poate realiza în mod temporar sau permanent.

Pentru a realiza o configurare temporară se pornește clientul de DHCP, de exemplu cu ajutorul comenzii `dhclient`:

```
asgard:/home/george# dhclient
Internet Systems Consortium DHCP Client V3.0.6
Copyright 2004-2007 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/

Listening on LPF/eth0/00:40:f4:cc:c5:41
Sending on   LPF/eth0/00:40:f4:cc:c5:41
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 6
DHCPOFFER from 10.1.1.1
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPACK from 10.1.1.1
bound to 10.1.1.2 -- renewal in 815 seconds.
```

Interfața va rămâne astfel configurată până la următoarea repornire a stației sau pană când configurările vor fi schimbate.

Pentru a realiza o configurare permanentă prin DHCP se folosește fișierul `/etc/network/interfaces`. Linia care configurează interfața `eth0` arată astfel:

```
iface eth0 inet dhcp
```

7.4.4 DNS. Configurarea serverelor de DNS

Pentru a configura adresele serverelor de DNS se folosește fișierul `/etc/resolv.conf`. În cadrul acestui fișier se pot specifica adresele mai multor servere de DNS. În cazul în care unul din ele nu conține informații despre domeniul căutat, se va încerca interogarea celorlalte.

Manualul de utilizare al fișierului `/etc/resolv.conf` se poate accesa folosind comanda:

```
man resolv.conf
```

Un exemplu de fișier `/etc/resolv.conf` este următorul:

```
search localdomain
nameserver 10.1.1.1
nameserver 217.115.138.24
nameserver 128.107.241.185
```

În cadrul fișierului cea mai importantă declarație este declarația `nameserver`. Ea definește adresa unui server de DNS. Fișierul poate conține mai multe declarații de acest tip.

Aplicarea configurărilor se realizează imediat deoarece fișierul `resolv.conf` este interogat de fiecare proces atunci când dorește rezolvarea unui nume de domeniu.

Deşi nu este uzual, serverele de DNS se pot configura şi din cadrul fișierului `/etc/network/interfaces`.

De cele mai multe ori, atunci când o interfață este configurată automat prin DHCP, serverul de DHCP oferă pe lângă adresa IP şi adresa unui server DNS, care va fi salvată automat în `/etc/resolv.conf`.

7.4.5 Aliasuri. /etc/hosts

Aliasurile reprezintă etichete care sunt asociate unor adrese de IP. Din punct de vedere principial, sistemul de aliasuri funcționează asemănător cu cel de DNS: de exemplu, atunci când o stație dorește să acceseze serverul numit `my-print-server`, ea va încerca să găsească adresa IP a serverului respectiv.

Există două deosebiri majore față de DNS. Prima constă în faptul ca etichetele asociate unor adrese IP nu trebuie să respecte convențiile numelor de domenii DNS (de exemplu nu trebuie să se termine cu un *Top Level Domain*). A doua deosebire constă în felul în care este găsită adresa IP asociată unei etichete. Dacă în cazul DNS se folosesc servere dedicate, în cazul aliasurilor este interogat fişierul `/etc/hosts`. În acest fişier sunt trecute corespondențele între etichete şi adrese IP.

Un exemplu de fişier `hosts` este următorul:

```
asgard:/home/george# cat /etc/hosts
127.0.0.1 localhost asgard
127.0.1.1 asgard

#Servers
10.1.1.100 print.mydomain.ro my-print-server
10.1.1.101 ftp.mydomainro my-ftp-server
192.168.1.150 mail-server

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Fişierul `hosts` conține mai multe linii de forma:

```
adresă_IP numele_stației [etichetă] [etichetă] ...
```

Câmpul cel mai din stânga este adresa de IP. Urmează apoi numele principal al stației, şi etichetele care se asociază cu această adresă, despărțite prin spațiu.

De obicei, atunci când sunt configurate şi servere de DNS şi aliasuri, aliasurile au prioritate şi vor fi verificate primele.

7.5 Testarea configurațiilor de rețea

Odată ce s-au realizat configurațiile de rețea, acestea trebuie testate. Pentru testarea lor sunt folosite în mod uzual două utilitare: ping şi traceroute. Aceste utilitare sunt folosite şi pentru a depista probleme uzuale care pot apărea în funcționarea unei rețele.

7.5.1 Ping

Utilitarul ping este folosit pentru a verifica conectivitatea între două stații. El testează conexiunea fizică şi parte din cea logică, folosindu-se de protocoalele IP şi ICMP (*Internet Control Message Protocol*) pentru a trimite şi pentru a primi mesaje.

Ping primeşte adresa IP a stației destinație şi va trimite către aceasta un mesaj ECHO REQUEST. Atunci când acest mesaj este recepționat, stația destinație va răspunde cu un mesaj similar, ECHO REPLY. Dacă răspunsul destinației ajunge înapoi la sursă, conexiunea dintre ele funcționează. În caz contrar există probleme care împiedică cele două stații că comunice.

Principalii parametri ai comenzii `ping` sunt următorii:

```
ping [-f] [-c count] [-i interval] [-I interface] destination
```

Fără nici o opțiune, `ping` va trimite un număr nelimitat de mesaje către destinație, până când va fi întrerupt:

```
asgard:/home/george# ping 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.395 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.359 ms
[CTRL+C]
--- 10.1.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.341/0.400/0.707/0.106 ms
```

Opțiunea `-c` specifică numărul de pachete trimise, iar opțiunea `-i` specifică intervalul în secunde între două pachete trimise:

```
asgard:/home/george# ping 10.1.1.1 -c 3 -i 1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.326 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.459 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=255 time=0.361 ms
--- 10.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.326/0.382/0.459/0.056 ms
```

Opțiunea `-I` specifică pe ce interfață se vor trimite pachetele:

```
asgard:/home/george# ping 10.1.1.1 -I eth0 -c 3
PING 10.1.1.1 (10.1.1.1) from 10.1.1.2 eth0: 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.331 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.326 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=255 time=0.367 ms

--- 10.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.326/0.341/0.367/0.023 ms
```

Opțiunea `-f` permite transmiterea unui număr nelimitat de pachete ECHO REQUEST, fără să se aştepte răspunsul ECHO REPLY. În acest fel destinația este inundată (*flood*) de mesaje. Datorită potențialului pe care îl are această opțiune de a fi folosită în atacuri DoS [vezi 9.4.1], parametrul `-f` poate fi folosit doar de către root:

```
asgard:/home/george# ping -f 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
.....................................................................
.........................
CTRL+C
--- 10.1.1.1 ping statistics ---
100 packets transmitted, 3 received, 97% packet loss, time 8148ms
rtt min/avg/max/mdev = 0.240/0.389/2.291/0.450 ms, ipg/ewma 12.053/0.364
ms
```

7.5.2 Traceroute

Utilitarul `traceroute` este folosit pentru a testa calea pe care o urmează pachetele între sursă şi destinaţie. El va afişa câte un mesaj pentru fiecare staţie/server/echipament de reţea prin care trec mesajele.

Pentru a realiza acest lucru traceroute se foloseşte de câmpul **TTL** (*Time To Live*) prezent în antetul fiecărui pachet. Din motive ce ţin de felul în care funcţionează o reţea este posibil ca în drumul între sursă şi destinaţie să apară bucle. Dacă un pachet intră într-o astfel de buclă, el va fi trimis la infinit între staţiile ce aparţin buclei şi nu va ajunge niciodată la destinaţie. Pentru a preveni o asemenea situaţie, a fost introdus câmpul TTL în cadrul antetului fiecărui pachet. Acest câmp conţine numărul maxim de echipamente de reţea prin care poate trece un pachet înainte de a fi aruncat.

Dacă un pachet are câmpul TTL cu valoarea iniţială (atunci când pleacă de la sursă) egală cu 10, atunci el poate trece prin maxim 10 echipamente de reţea. Al zecelea echipament va arunca pachetul şi va trimite un mesaj ICMP sursei prin care o va înştiinţa de acest lucru. Trimiţând acest mesaj, al zecelea echipament îşi va anunţa identitatea către sursa pachetului.

Pe acest sistem se bazează funcţionarea utilitarului traceroute. Se trimit pachete ce au câmpul TTL configurat la o valoare din ce în ce mai mare. Atunci când valoarea TTL este 1, va răspunde primul echipament de reţea de pe cale şi va spune că a aruncat pachetul trimis. Atunci când valoarea TTL este 2, va răspunde al doilea echipament de pe cale şi va spune că a aruncat pachetul trimis. În felul aceasta toate echipamentele de pe parcurs vor fi identificate unul câte unul.

Parametrii uzuali ai comenzii `traceroute` sunt următorii:

```
traceroute [-f first_ttl] [-m max_ttl] host
```

`host` reprezintă destinaţia până la care se doreşte aflarea rutei:

```
asgard:/home/george$ traceroute rol.ro
traceroute to rol.ro (195.95.229.171), 30 hops max, 40 byte packets
 1  10.1.1.1 (10.1.1.1)  0.429 ms  0.332 ms  0.248 ms
 2  10.100.0.1 (10.100.0.1)  0.809 ms  0.657 ms  0.527 ms
 3  backbone.hertza.ro (89.165.146.177)  6.286 ms  1.765 ms  2.493 ms
 4  84.234.111.41 (84.234.111.41)  6.738 ms  7.507 ms  11.869 ms
 5  84.234.107.154 (84.234.107.154)  4.406 ms  1.603 ms  1.625 ms
 6  195.95.229.171 (195.95.229.171)  2.715 ms  1.676 ms  1.230 ms
```

Pentru fiecare echipament de pe parcurs este afişată adresa lui şi timpul de răspuns pentru trei încercări succesive. În cazul în care pentru un mesaj trimis nu este primit răspuns, se va afişa caracterul *.

Parametrul `-f` permite configurarea valorii câmpului TTL din primul mesaj trimis. Valoarea câmpului TTL din mesajele următoare va depinde de aceasta şi se va incrementa cu 1 la fiecare mesaj.

```
asgard:/home/george$ traceroute -f 4 rol.ro
traceroute to rol.ro (195.95.229.171), 30 hops max, 40 byte packets
 4  84.234.111.41 (84.234.111.41)  2.008 ms  8.286 ms  10.525 ms
 5  84.234.107.154 (84.234.107.154)  3.399 ms  4.614 ms  5.497 ms
 6  195.95.229.171 (195.95.229.171)  4.458 ms  2.901 ms  2.947 ms
```

Parametrul `-m` configurează valoarea maximă a câmpului TTL care va fi încercată în cazul în care destinația nu a fost atinsă. Dacă paramerul nu este specificat, se vor testa implicit cai cu lungimi de până la 30 de echipamente.

```
asgard:/home/george$ traceroute -m 3 rol.ro
traceroute to rol.ro (195.95.229.171), 4 hops max, 40 byte packets
 1  10.1.1.1 (10.1.1.1)  0.404 ms  0.402 ms  0.326 ms
 2  10.100.0.1 (10.100.0.1)  0.623 ms  8.407 ms  2.069 ms
 3  backbone.hertza.ro (89.165.146.177)  9.707 ms  6.819 ms  7.895 ms
```

7.5.3 Cum se depistează problemele uzuale în cazul configurărilor de rețea

Atunci când apar probleme în configurările de rețea, pentru depistarea lor există două abordări: de sus în jos (*top-down*) şi de jos în sus (*bottom-up*). Partea de sus (*top*) a unei rețele se referă la partea de aplicații, iar partea de jos (*bottom*) se referă la conexiunile fizice între echipamente.

În continuare va fi analizată abordarea de jos în sus, deoarece majoritatea problemelor dintr-o rețea sunt cauzate de conexiunile fizice între echipamente.

Să presupunem că nu poate fi accesată pagina web www.google.ro. În cazul acestei abordări primul lucru care trebuie verificat este faptul că există legătură fizică între stație şi switch sau punctul de acces fără fir. În cazul legăturilor la Ethernet (realizate prin cablu UTP) conectivitatea se poate verifica examinând led-ul de link de pe placa de rețea. Dacă led-ul este aprins, există conexiune între stație şi switch. Dacă led-ul este stins, trebuie verificat cablul de rețea sau funcționarea switch-ului.

O altă metodă folosită pentru a verifica starea link-ului pentru o interfață de rețea, utilă în special atunci când nu există acces fizic la stație, este utilizarea unui program de genul `ethtool`. Printre altele, acesta oferă informații detaliate despre tipul de legătură fizică existentă şi starea ei. Pentru a vedea starea unei interfețe utilitarul se apelează având ca parametru numele interfeței:

```
asgard:/home/george# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 32
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: d
        Current message level: 0x00000007 (7)
        Link detected: yes
```

În cazul în care `Link detected` are valoarea `yes`, există conexiune între stație şi switch. Dacă are valoarea `no`, trebuie verificat cablul de rețea sau funcționarea switch-ului.

Următorul pas este verificarea existenței unui gateway şi a conexiunii cu acesta. Pentru a verifica faptul că un gateway a fost configurat trebuie analizat fişierul

`/etc/network/interfaces` sau rezultatul comenzii `route`, funcție de configurație. Dacă niciun gateway nu este configurat, trebuie configurat unul. Dacă gateway-ul este configurat, trebuie verificată legătura către acesta cu ajutorul comenzi ping.

După verificarea gateway-ului, trebuie verificat DNS-ul. Pentru aceasta trebuie analizat fișierul `/etc/resolv.conf`. În cazul în care nu este configurat niciun server DNS, trebuie configurat unul. Dacă este configurat un server, trebuie testată legătura cu acesta cu ajutorul comenzii ping.

Pasul următor este verificarea conectivității cu un alt server din Internet. Se poate încerca accesarea unei alte pagini web, se poate testa conectivitatea cu un alt server cu ajutorul comenzii ping sau se poate încerca verificarea căii către google.ro cu ajutorul comenzii traceroute. Folosirea traceroute oferă avantajul depistării echipamentului de rețea în care se opresc pachetele.

7.6 Studii de caz

7.6.1 Verificarea în linie de comandă a parametrilor de rețea în Windows

Cea mai ușoară metodă de a verifica parametrii de rețea în Windows este folosirea utilitarului `ipconfig`. Acesta se apelează din linie de comandă. Pentru a porni interpretorul de comenzi din Windows XP, se merge la Start→Run, unde în câmpul *Open* se introduce `cmd` și se apasă pe OK.

Informații despre modul de utilizare al comenzii `ipconfig`, precum și despre parametrii disponibili, se pot obține utilizând parametrul `/?`:

```
C:\Documents and Settings\George >ipconfig /?
```

Apelarea comenzii `ipconfig` fără nici un parametru va afișa parte din parametrii de rețea:

```
C:\Documents and Settings\George>ipconfig
Windows IP Configuration

Ethernet adapter Local Area Connection:
        Media State . . . . . . . . . . . : Media disconnected

Ethernet adapter Wireless Network Connection 2:

        Connection-specific DNS Suffix  . : lan
        IP Address. . . . . . . . . . . . : 192.168.1.70
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.254
```

Pentru a vedea toți parametrii de rețea, inclusiv serverul DNS și gateway-ul implicit, se folosește parametrul `/all`:

```
C:\Documents and Settings\George>ipconfig /all
Windows IP Configuration
        Host Name . . . . . . . . . . . . : Asgard
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Mixed
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
        DNS Suffix Search List. . . . . . : lan

Ethernet adapter Local Area Connection:
        Media State . . . : Media disconnected
        Description . . . : Broadcom 440x 10/100 Integrated Controller
        Physical Address. : 00-40-F4-CC-C5-41

Ethernet adapter Wireless Network Connection 2:
        Connection-specific DNS Suffix  . : lan
        Description . . . . : Atheros AR5006X Wireless Network Adapter
        Physical Address. . . . . . . . . : 00-19-7E-0B-7E-65
        Dhcp Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : 192.168.1.70
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.254
        DHCP Server . . . . . . . . . . . : 192.168.1.254
        DNS Servers . . . . . . . . . . . : 192.168.1.254
        Lease Obtained. . . . . . . . . . : 11 august 2007 16:17:26
        Lease Expires . . . . . . . . . . : 12 august 2007 16:17:26
```

Configurarea parametrilor de rețea din line de comandă în Windows este mai dificil de realizat decât în Linux. Pentru a putea face acest lucru se foloseşte utilitarul `netsh`. Informații despre modul de utilizare al acestuia se pot obține folosind parametrul `/?`:

```
C:\Documents and Settings\George >netsh /?
```

7.6.2 Configurări de rețea în Windows XP

Cel mai uzual şi mai simplu mod de a configura parametrii de rețea în Windows XP este folosirea utilitarelor din interfața grafică. Pentru aceasta se accesează *Control Panel*, de unde se selectează *Network Connections*:

Figura 7-4 Selectarea Network Cnnections din Control Panel

După accesarea *Network Connections* va fi afişată o fereastră unde este prezent câte un simbol pentru fiecare interfață de rețea:

Figura 7-5 Simbolutile asociate interfețelor de rețea

Dacă este selectat unul din simboluri, în partea stângă a ecranului vor fi afişate configurările principale ale interfeței de rețea corespunzătoare:

Figura 7-6 Configurările principale ale unei interfețe de rețea

Pentru a putea vedea starea interfeței şi pentru a putea configura interfața trebuie afişat meniul contextual corespunzător ei. Acest lucru se realizează efectuând cu mouse-ul un click dreapta pe simbolul asociat interfeței:

Figura 7-7 Meniul contextual corespunzător unei interfețe de rețea

Intrările prezente în acest meniu sunt:

- *Disable*, pentru dezactivarea interfeței. Este corespunzător opțiunii `down` a comenzii `ifconfig` din Linux
- *View Available Wireless Networks* (care apare doar în cazul interfețelor fără fir) permite afişarea unei liste de rețele fără fir disponibile şi alegerea uneia din ele
- *Status* (care este si opțiunea ce se va apela în cazul în care se efectuează cu mouse-ul un dublu click stânga pe simbolul asociat interfeței de rețea) permite afişarea unei ferestre ce prezintă starea curentă a interfeței şi a legăturii cu switch-ul sau punctul de acces fără fir:

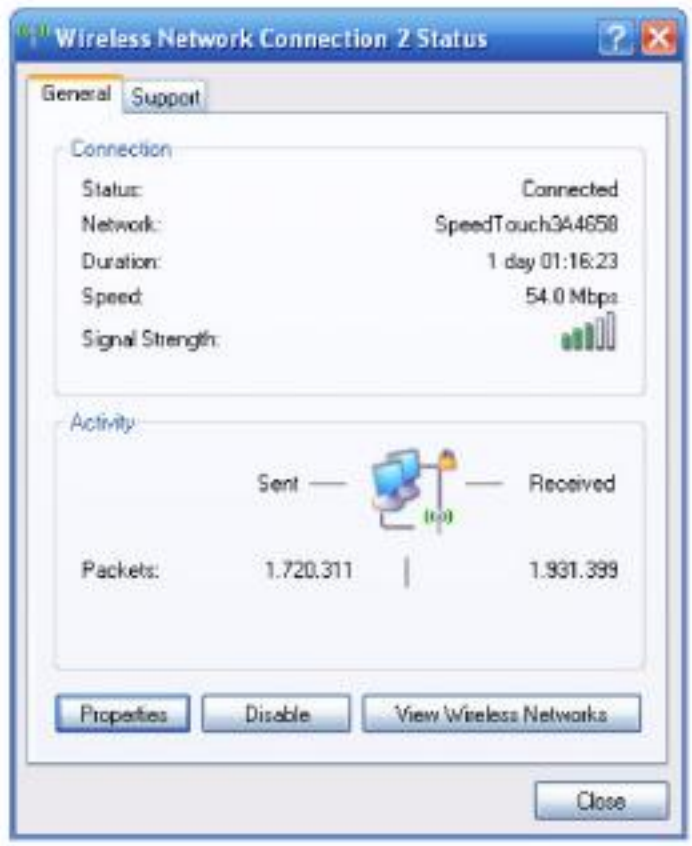


Figura 7-8 Starea unei interfețe de rețea

- *Repair* va încerca reconfigurarea interfeței
- *Bridge Connections* permite realizarea unei legături între interfețe pentru a transfera pachete de pe una pe cealaltă. În acest fel stația se va comporta într-un mod asemănător cu un switch.
- *Poperties* permite afişarea unei ferestre de configurare a interfeței:

Figura 7-9 Proprietățile unei interfețe de rețea

Pentru a configura adresa IP a interfeței, gateway-ul şi serverul DNS dintre itemii folosiți de conexiune, prezentați în cadrul unei liste în centrul ferestrei, se selectează ultimul, *Internet*

Protocol (TCP/IP), şi se apasă pe butonul *Properties*. Va fi afişată încă o fereastră ce va permite configurare adresei IP, a măştii de reţea, a gateway-ului implicit şi a serverului DNS.

Figura 7-10 Configurarea DHCP pentru o interfaţă de reţea

Pentru a configura interfaţa folosind DHCP, se aleg opţiunile *Obtain an IP address automatically* şi *Obtain DNS server address automaticaly*.

Pentru a configura manual adresele, se selectează *Use the following IP address* şi *Use the following DNS server addresses* şi se introduc adresele în câmpurile corespunzătoare:

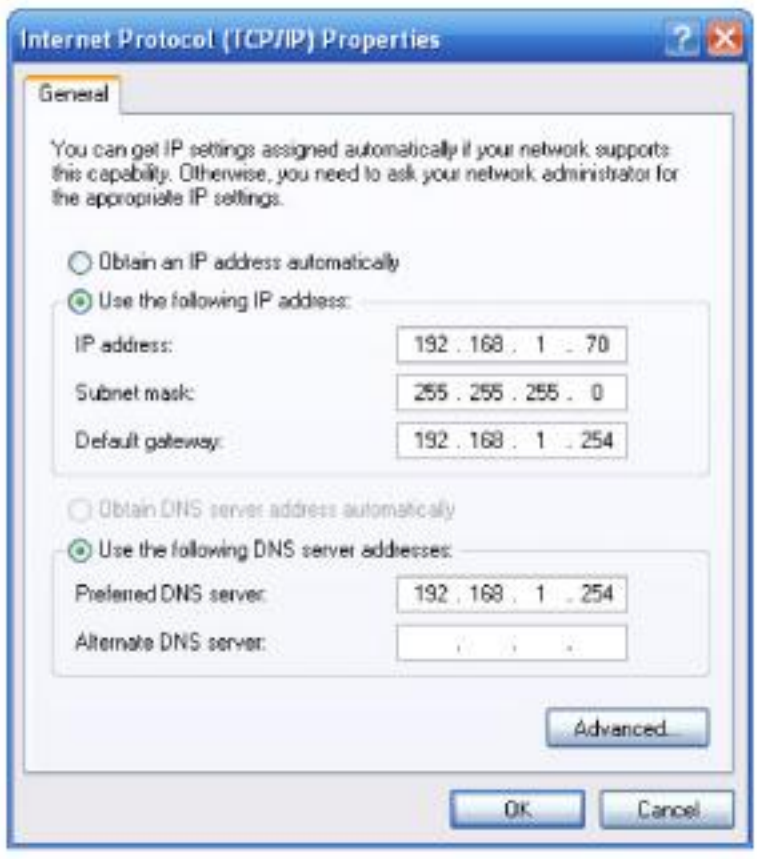


Figura 7-11 Configurarea manuală a parametrilor de reţea

7.6.3 Configurări de rețea în Windows Vista

Modul de configurare al unei interfețe de rețea în Windows Vista este asemănător cu cel din Windows XP. Diferența importantă între ele este dată de modul în care se accesează fereastra de configurare a parametrilor.

Din meniul Control Panel se alege *Network and Internet*:

Figura 7-12 Selectarea Network and Internet din Control Panel

Apoi se selectează *Network Sharing Center*:

Figura 7-13 Selectarea Network Sharing Center

Din noua fereastră se alege *Manage Network Connections*:

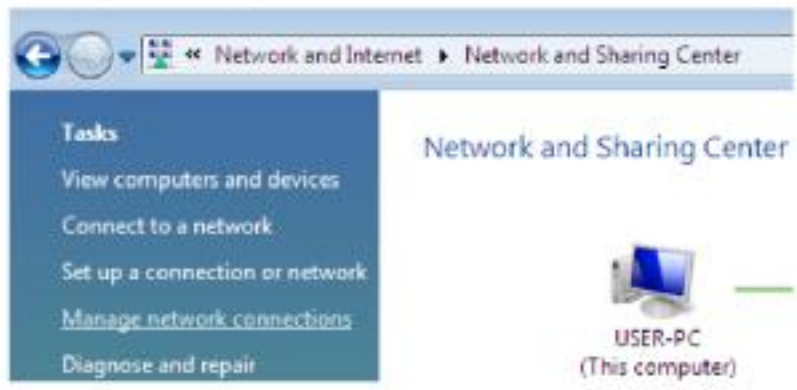


Figura 7-14 Selectarea Manage Network Connections

Urmând ca apoi să fie afișată o listă cu interfețele de rețea disponibile. Prin efectuarea unui click dreapta pe simbolul asociat unei interfețe va apărea un meniu contextual asemănător cu cel din Windows XP:

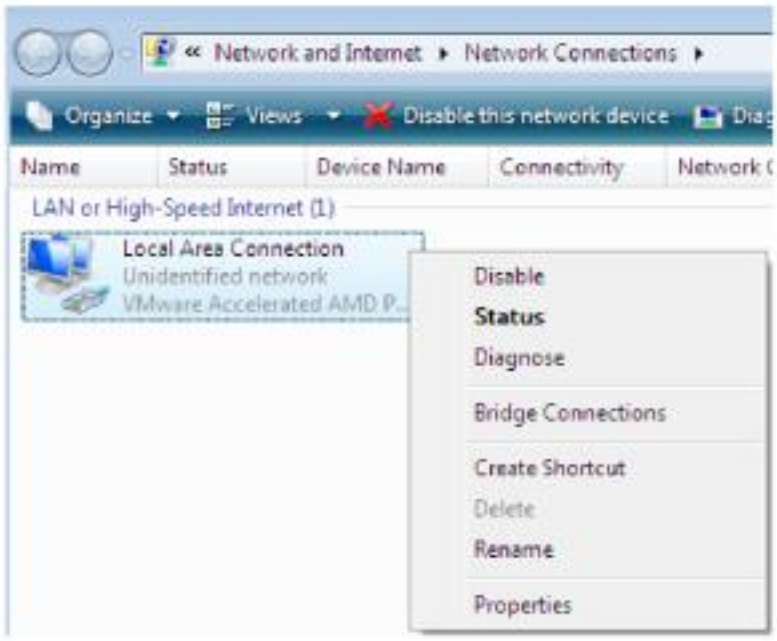


Figura 7-15 Meniul contextual corespunzător unei interfețe de rețea

Prin selectarea opțiunii *Properties* din acest meniu va fi afişată fereastra de configurare a parametrilor interfeței:

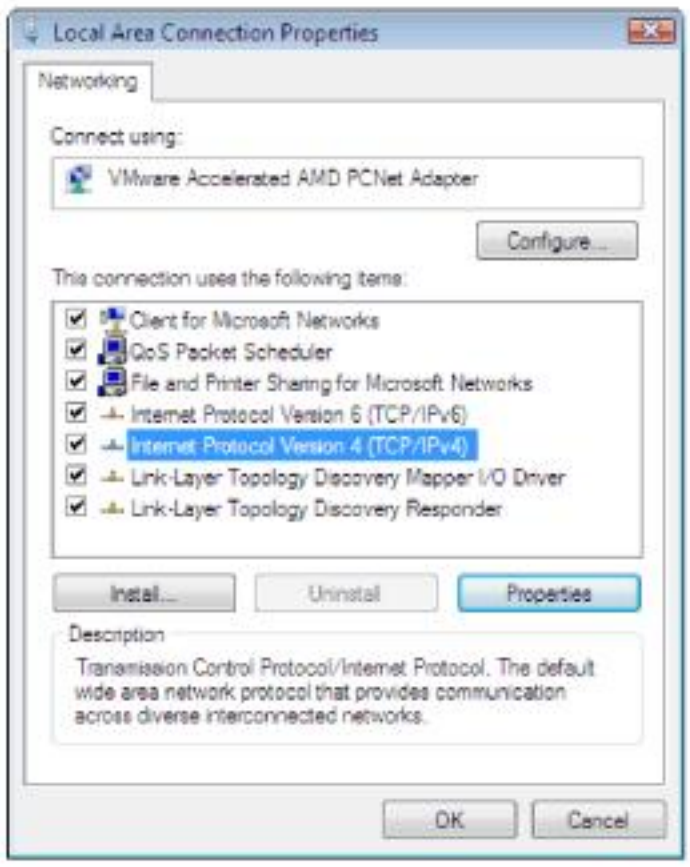


Figura 7-16 Proprietățile unei interfețe de rețea

Pentru a configura adresa IP a interfeței, gateway-ul şi serverul DNS, se alege itemul *Internet Protocol Version 4*.

Fereastra de configurare a parametrilor este identică celei din Windows XP:

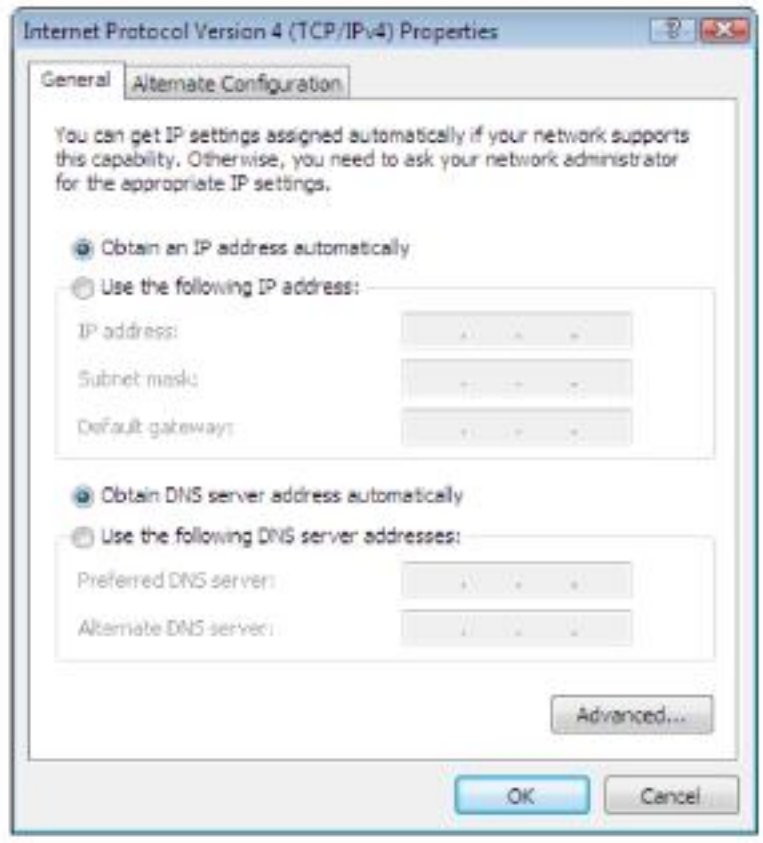


Figura 7-17 Configurarea parametrilor de rețea

În cazul în care este aleasă opțiunea *Obtain an IP address automatically*, adresa IP a interfeței va fi configurată prin DHCP. Pentru a specifica manual parametrii interfeței trebuie aleasă opțiunea *Use the following IP address.*

7.6.4 Configurarea PPPoE în Linux

PPP (*Point to Point Protocol*) este un protocol utilizat pentru comunicația punct-la-punct (comunicația dintre două noduri conectate direct între ele). PPP permite realizarea de legături peste conexiuni fizice seriale, lini telefonice, conexiuni radio sau fibră optică.

Dintre implementările PPP, cele mai uzuale sunt cele folosite de conexiunile de tip dial-up şi PPPoE.

PPPoE (*PPP over Ethernet*) este un protocol care permite încapsularea (includerea) cadrelor PPP în cadre Ethernet, cel mai frecvent pentru a putea fi trimise printr-o placa de rețea obişnuită (ce foloseşte cablu UTP).

PPPoE este folosit uzual în cadrul serviciilor de tip ADSL sau Internet prin cablu TV. Dacă legătura fizică dintre calculator şi modem se realizează prin cablu UTP, atunci este foarte probabil ca legătura logică dintre calculator şi modem să folosească PPPoE. Pentru a putea să acceseze Internetul, utilizatorul trebuie mai întâi să stabilească o conexiune PPP cu modemul.

Stabilirea unei conexiuni PPP necesită, de cele mai mule ori, autentificarea utilizatorului. Aceasta se realizează pe baza unui nume de utilizator şi a unei parole furnizate de către ISP (*Internet Service Provider* – furnizorul de servicii de Internet).

Pentru configurarea parametrilor conexiunii PPP, inclusiv a parametrilor necesari autentificării, cea mai uşoară metodă este folosirea utilitarului `pppoeconf`. Instalarea acestuia depinde de instalarea tuturor pachetelor necesare funcționării protocolului PPP, astfel încât, pe o distribuție ce foloseşte pachete cum este Ubuntu sau Debian, o dată cu `pppoeconf` se vor instala si toate dependințele sale.

Pe de altă parte, odată configurată conexiunea PPP cu ajutorul `pppoeconf`, acesta se va ocupa de realizarea tuturor fişierelor de configurare, inclusiv de adăugarea unui script care la pornirea sistemului de operare va realiza conectarea prin PPP.

Pentru configurarea PPPoE sunt necesare următoarele fişiere (ele sunt create sau actualizate automat de către `pppoeconf`):

- `/etc/network/interfaces`
- `/etc/ppp/pap-secrets`
- `/etc/ppp/peers/dslprovider`

Ultimul dintre fişiere poate avea orice nume şi se poate afla în orice loc, însă trebuie specificat în cadrul fişierului `/etc/network/inerfaces`.

Un exemplu de `etc/network/interfaces` este următorul:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet manual

auto dsl-provider
iface dsl-provider inet ppp
pre-up /sbin/ifconfig eth0 up # line maintained by pppoeconf
provider dsl-provider

auto eth0
```

Autentificarea conexiunii PPP se realizează folosind protocolul PAP (*Password Authentication Protocol*), fie CHAP (*Challenge-Handshake Authentication Protocol*). Un exemplu de fişier `/etc/ppp/pap-secrets`, în care sunt trecute informaţiile necesare autentificării, este următorul:

```
[...]

# INBOUND connections

# Every regular user can use PPP and has to use passwords from /etc/passwd
*       hostname        ""      *

# UserIDs that cannot use PPP at all. Check your /etc/passwd and add any
# other accounts that should not be able to use pppd!
guest   hostname        "*"     -
master  hostname        "*"     -
root            hostname        "*"     -
support hostname        "*"     -
stats   hostname        "*"     -

# OUTBOUND connections

# Here you should add your userid password to connect to your providers via
# PAP. The * means that the password is to be used for ANY host you connect
# to. Thus you do not have to worry about the foreign machine name. Just
# replace password with your password.
# If you have different providers with different passwords then you better
# remove the following line.
#       *       password

"student" * "usorules"
```

În fişierul `/etc/ppp/peers/dslprovider` sunt trecute informaţii legate de parametrii conexiunii PPP, alţii decât cei de autentificare:

```
# Minimalistic default options file for DSL/PPPoE connections

noipdefault
defaultroute
replacedefaultroute
hide-password
noauth
persist
#mtu 1492
#persist
#holdoff 20
plugin rp-pppoe.so eth0
user "student"
usepeerdns
```

Pentru a activa o conexiune PPP se foloseşte comanda `pon`. Ea este echivalentă cu parametrul `up` al comenzii `ifconfig`. Pentru a dezactiva o conexiune PPP se foloseşte comanda `poff`, echivalentă cu parametrul `down` al comenzii `ifconfig`.

Un amănunt important de precizat în legătură cu protocolul PPPoE este acela că, pentru a funcționa prin PPPoE, o interfață de rețea Ethernet nu necesită configurarea unei adrese IP. Interfața de rețea Ethernet este folosită indirect pentru conectarea la Internet: prin ea sunt trimise datele către modem acesta asigurând mai departe legătura către Internet.

În acest caz adresa IP este asociată cu o interfața virtuală numită ppp0:

```
asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet6 addr: fe80::240:f4ff:febb:1701/64 Scope:Link
          [...]

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

ppp0      Link encap:Point-to-Point Protocol
          inet addr:192.168.10.3  P-t-P:10.0.0.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1492  Metric:1
          RX packets:176 errors:0 dropped:0 overruns:0 frame:0
          TX packets:180 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:105255 (102.7 KiB)  TX bytes:13991 (13.6 KiB)
```

După cum se poate observa, interfața `eth0` nu are configurată adresa IP. În schimb interfața `ppp0`, asociată conexiunii PPP, este cea care fost configurată cu adresa IP 192.168.10.3.

Cuvinte cheie

- rețea de calculatoare
- LAN, MAN, WAN
- protocolul IP; adresă IP
- adresă MAC
- Ethernet
- UTP
- router
- unicast, multicast, broadcast
- adresă de (sub)rețea
- default gateway
- DNS
- switch
- Access Point
- DHCP
- configurare temporară/permanentă;
- configurare statică/dinamică
- ifconfig
- /etc/network/interfaces
- /etc/resolv.conf
- /etc/hosts
- route
- DHCP
- dhclient
- ping
- traceroute
- ipconfig
- PPPoE

Întrebări

Un utilizator încearcă să testeze conectivitatea către google.ro cu ajutorul comenzii ping: "ping cs.pub.ro" si testul reuşeşte. Totuşi, dacă utilizatorul tastează în bara de adrese a browser-ului "cs.pub.ro", pagina web respectivă nu poate fi accesată. Care poate fi motivul pentru care cele două teste au rezultate diferite?

- ❑ Serverul web de la adresa cs.pub.ro este oprit
- ❑ Utilizarea comenzii ping este greşită
- ❑ Comanda ping a oprit serverul web
- ❑ Utilitarul ping nu oferă informații despre conexiunea cu cs.pub.ro

Utilizatorul Dorel încearcă să se conecteze de acasă prin ssh pe una din stațiile de la firma la care lucrează, prin comanda "ssh dorel@10.183.257.135". Conexiunea nu se stabileşte. Care este motivul pentru care Dorel nu se poate conecta pe stația de la firmă?

- ❑ Nu este configurat un server DNS
- ❑ Nu este configurat un gateway
- ❑ Comanda prin care încearcă să se conecteze este greşită
- ❑ Nici una din variantele de mai sus

Dându-se o stație care are configurată adresa IP 172.16.150.200 şi masca de rețea 255.255.255.0, care este adresa rețelei din care face parte stația?

- ❑ 172.0.0.0
- ❑ 172.16.0.0
- ❑ 172.16.150.0
- ❑ 172.16.150.200

Deşi legătura la Internet funcționa, utilizatorul Dorel a dorit să testeze conectivitatea cu gateway-ul cu ajutorul comenzii „ping -f 192.168.10.1". La scurt timp el a fost contactat de către administratorul de rețea care l-a rugat să îsî verifice calculatorul impotriva virușilor. Acest lucru s-a petrecut deoarece:

- ❑ Dorel foloseste Linux şi nu Windows, la fel ca majoritatea celorlalți utilizatori
- ❑ Dorel nu a repornit calculatorul de mult timp
- ❑ Dorel a generat foarte mult trafic în timp scurt, lucru specific viruților
- ❑ Dorel nu obisnuieşte să cripteze email-urile trimise.

Rolul serviciului DNS este de a:
- Asocia nume de domenii cu adrese MAC
- Asocia nume de domenii cu adresa IP
- Asocia adrese IP cu adrese MAC
- Asocia adresa IP cu numele utilizatorului

În fişierul "/etc/network/interfaces" se găseşte următoarea linie: "iface eth0 inet dhcp". În linia de comandă este dată următoarea comandă: "# ifconfig eth0 192.168.124.150", după care staţia este reporită. Care va fi adresa IP a interfeţei eth0 imediat după repornirea staţiei?
- 192.168.124.150 pentru că a fost ultima adresă configurată de către root
- 192.168.124.150 dacă aceasta este adresa furnizată staţiei de către serverul DHCP
- Interfaţa nu va avea nici o adresă IP pentru că s-a creat un conflict prin configurarea adresei IP în două moduri
- Interfaţa nu va avea nici o adresă IP pentru că nu a fost dat parametrul netmask comenzii ifconfig

Configurarile permanente se realizează cu ajutorul fişierului /etc/network/ip. Configurările temporare îşi pierd efectul o dată cu reornirea sistemului.
- Adevărat, fals
- Adevărat, adevărat
- Fals, fals
- Fals, adevărat

Masca de rețea este utiliztă pentru a identifica adresa rețelei din cadrul unei adrese IP. Pentru a putea accesa serverele din Internet este necesara configurarea unui server DNS
- Adevărat, fals
- Adevărat, adevărat
- Fals, fals
- Fals, adevărat

Cu ajutorul comenzii traceroute se poate testa:
- Lăţimea de bandă a unei conexiuni
- Calea dintre sursă şi destinaţie
- Funcţionarea serverului web de pe staţia destinaţie
- Existenţa unui ruter între sursă şi destinaţie

Dacă un server DNS nu este specificat, care dintre următoarele comenzi este probabil să funcţioneze:
- ping www.google.ro
- ping cs.pub.ro
- ping localhost
- ping dns.localdomain.ro

8 Servicii de rețea

"When I took office, only high energy physicists had ever heard of what is called the World Wide Web.... Now even my cat has its own page."
Bill Clinton, 1996

Ce se învață din acest capitol?

- Conceptul de protocol şi stivă de protocoale
- Modelul client-server
- Folosirea porturilor
- Execuția comenzilor la distanță: telnet, ssh, wget, curl
- Funcționarea serviciului de poştă electronică (email)
- Funcționarea serviciului de web
- Modalități de partajare a fişierelor: SMB, FTP, rețele de partajare

8.1 Concepte specifice aplicațiilor de rețea

8.1.1 Stiva TCP/IP

Sistemele de calcul au început să fie dezvoltate odată cu sfârşitul celui de al doilea război mondial. Cu toate acestea problematica asigurării comunicației între sisteme aflate în locații la mare distanță a fost formulată abia spre începutul anilor `70, de către US DoD (US *Department of Defense* - Ministerul Apărării Naționale al Statelor Unite). Proiectul US DoD a creat şi un cadru de standardizare pentru soluțiile de comunicație, denumit stiva de protocoale TCP/IP (*Transmission Control Protocol/Internet Protocol*).

*Un **protocol** este un set de reguli care guvernează modul în care două dispozitive schimbă informație într-o rețea şi asigură coerența comunicației.*
*O **stivă de protocoale** reprezintă un set de protocoale ce comunică între ele prin primitive (funcții) clar definite.*

O stivă de protocoale este în general privită ca o succesiune de niveluri. Fiecare dintre niveluri este descris de:

- un set de funcții pe care le îndeplineşte,
- un set de parametrii pe care îi aşteaptă de la nivelul inferior
- serviciile pe care le oferă nivelului imediat superior.

Protocoalele dintr-o stivă nu vor putea comunica între ele direct decât dacă se află pe niveluri adiacente.

Descrierea unei soluții de comunicare prin prisma unei stive de protocoale oferă în primul rând avantajul flexibilității procesului de standardizare. Astfel, schimbarea (sau chiar simpla optimizare) unui protocol din stiva de protocoale poate fi făcută fără a solicita modificări în restul protocoalelor.

Începuturile rețelelor de date au fost marcate de existența a multe protocoale de comunicație, dezvoltate în paralel, de vendori diferiți. Problema consta în faptul că deşi două protocoale, de la doi vendori diferiți, descriau amândouă foarte bine specificațiile propriilor dispozitive, nu puteau să comunice unul cu celălalt. Astfel, pentru a face posibilă intercomunicarea între echipamentele de rețea produse de vendori diferiți, s-a introdus conceptul de standardizare a unui protocol. Odată ce un protocol este acceptat ca standard, el devine dintr-un set de reguli, un **set comun de reguli**, fiind recunoscut din punct de vedere procedural şi funcțional de toți vendorii.

Pentru a descrie prelucrările efectuate asupra datelor trimise de la utilizator spre mediul de transmisie, stiva TCP/IP identifică patru niveluri: Aplicație, Transport, Internet (sau Rețea) şi Acces la mediu în figura de mai jos fiind prezentată stiva TCP/IP.

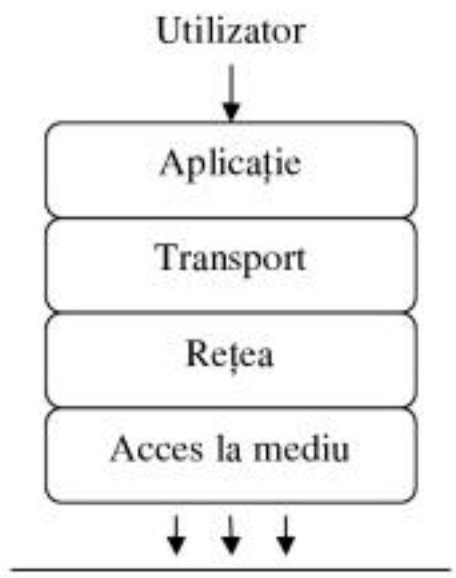


Figura 8-1 Stiva de protocoale TCP/IP

Nivelul aplicație este cel mai apropiat de utilizator şi oferă acestuia modalitatea de accesare a serviciului de rețea. Un serviciu de rețea poate fi implementat prin unul sau mai multe protocoale de nivel aplicație.

Protocoalele de la acest nivel vor conține reguli referitoare la interacțiunea cu utilizatorul, formatarea şi reprezentarea datelor, precum şi la controlul dialogului între componentele serviciului.

În funcție de serviciul oferit, protocoalele de nivel aplicație pot fi grupate în:

- serviciul de transfer al fişierelor poate fi implementat folosind FTP (*File Transfer Protocol*), HTTP (*HyperText Transfer Protocol*), etc.
- serviciul de poştă electronică se bazează pe următoarele protocoale: SMTP (*Simple Mail Transfer Protocol*), POP3 (*Post Office Protocol 3*), IMAP (*Internet Mail Access Protocol*)
- serviciul de executarea comenzilor la distanță: telnet, ssh, rlogin
- alte protocoale: DNS (*Domain Name System*), DHCP (*Dynamic Host Configuration Protocol*), SNMP (*Simple Network Management Protocol*)

Nivelul transport este responsabil cu probleme legate de siguranță, control al fluxului şi corecție de erori. Tot la nivelul transport este realizată multiplexarea comunicației prin specificarea porturilor sursă şi destinație [vezi 8.1.3]. Cele mai importante protocoale de nivel transport sunt TCP (*Transmission Control Protocol)* şi UDP (*User Datagram Protocol*).

TCP este un **protocol orientat conexiune**, adică înainte de a trimite datele va negocia parametrii de comunicație între entitățile participante la comunicație, iar apoi pe tot parcursul

va asigura reordonarea pachetelor ajunse în ordine greşită la destinaţie, sau retransmisia celor pierdute.

Pentru a înţelege mai bine noţiunea de protocol orientat pe conexiune, se poate face o analogie cu transmiterea datelor prin reţeaua telefonică. În momentul efectuării unei sesiuni telefonice, se stabileşte o conexiune dedicată prin reţea între sursa şi destinatie, înainte de începerea comunicării. Deasemenea acest exemplu relevă faptul că într-un protocol orientat pe conexiune, destinaţia este mereu contactată înainte de începerea transmisiei propriu-zise (în analogia prin soneria telefonului)

UDP este un **protocol neorientat conexiune**, astfel UDP nu se va preocupa de iniţierea sau terminarea conexiunii, de pierderea pachetelor sau doar a succesiunii acestora. UDP va începe transmisia fără a verifica măcar dacă destinaţia finală va accepta respectivul trafic.

Deşi 95% din traficul Internet foloseşte protocolul TCP, pentru reţelele locale numeroase servicii se bazează pe UDP: DNS, SNMP, TFTP. Motivul principal îl reprezintă gradul redus al erorilor în reţelele locale (spre deosebire de WAN). UDP este mai simplu de implementat, dar şi mai eficient decât TCP: antetul (informaţia suplimentară adăugată de fiecare protocol) UDP este doar de 8 octeţi, faţă de 20 de octeţi cât are TCP.

Nivelul reţea (Internet) se ocupă cu transmiterea pachetelor de la o sursă la o destinaţie precizată, indiferent în ce reţea se află aceasta. Alegerea căii pachetului prin reţelele intermediare, comutarea lui dintr-o o reţea în alta şi optimizarea traseelor alese sunt responsabilitatea protocolului de reţea.

Nivelul acces la mediu se ocupă cu toate problemele legate de transmiterea efectivă a unui pachet pe o legătură fizică, incluzând şi aspectele legate de tehnologii şi de medii de transmisie.

8.1.2 Implementarea parţială a stivei TCP/IP

Dispozitivele de interconectare asigură transferul cât mai rapid al datelor în general pe baza unor algoritmi de decizie. Un dispozitiv de interconectare poate în acelaşi timp asigura conversia între două tipuri de reţea (două protocoale) diferite.

Criteriul fundamental de evaluare al eficienţei unui dispozitiv de interconectare este numărul de pachete comutate pe secundă. În funcţie de complexitatea deciziei de comutare numărul de pachete prelucrate scade. Pentru echipamentele de reţea ce folosesc un număr redus de criterii pentru decizie, se renunţă la implementarea nivelurilor superioare din stiva de protocoale TCP/IP.

O primă clasă de dispozitive de interconectare îşi bazează deciziile de comutare doar pe informaţiile protocoalelor de nivel acces la mediu. În această categorie intră switchurile, dar şi AP-urile (*access point*).

A doua clasă de dispozitive va analiza atât informaţiile de nivel acces la mediu, cât şi pe cele de nivel reţea, procesul de decizie fiind mai complex, dar şi mai lent decât în primul caz. Cel mai important dispozitiv de interconectare de nivel reţea este ruterul.

Dispozitivele de interconectare pot implementa doar o parte a stivei de protocoale. Spre exemplu un ruter nu trebuie să înteleagă conţinutul unui pachet de date pentru a putea lua decizia de trimitere pe calea optimă. Astfel, ruterul are nevoie doar de informaţiile de nivel acces la mediu şi nivel reţea şi va implementa doar primele două niveluri ale stivei TCP/IP.

Figura 8-2 Comunicaţia între două staţii aflate în reţele diferite

Se consideră o topologie simplă în care un client de web aflat pe stația A acceseaza un server aflat pe stația B. Cele două stații se află în rețele locale diferite conectate prin ruter. Rețeua din stânga este o rețea Ethernet, în vreme ce în rețeua din dreapta AP-ul (acces point) este conectat pe Ethernet cu ruterul R şi oferă conectivitate fară fir pentru stația B.

Modelul de comunicație poate fi descris generic folosind stiva de protocoale TCP/IP precum în Figura 8-3.

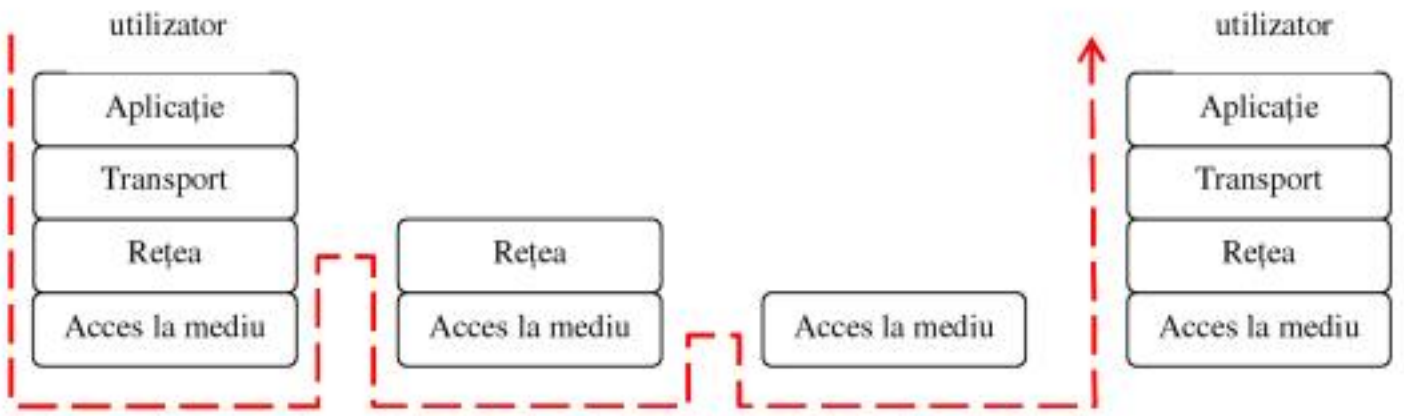


Figura 8-3 Descrierea generică a comunicației

Pentru topologia considerată mai sus la nivelul aplicație se foloseşte serviciul de web bazat pe protocolul HTTP. La nivelul transport şi nivelul rețea se foloseşte TCP, respectiv IP. Spre deosebire de nivelul aplicație unde putem avea o largă diversitate de protocoale folosite, la nivelurile transport şi rețea în Internet peste 95% din trafic este trafic TCP/IP.

Pentru nivelul acces la mediu în rețelele locale există două tehnologii dominante: Ethernet (IEEE 802.3) şi WLAN (*Wireless LAN*, IEEE 802.11). Conectivitatea între cele două standarde poate fi realizată ori de ruter (un ruter cu o interfață 802.11), sau de un AP (*acces point*).

Un AP nu are nevoie să interpreteze informația aflată la nivelul rețea pentru a realiza comutarea între rețeaua Ethernet şi rețeaua fără fir.

Cele două dispozitive de interconectare (ruterul şi AP-ul) au fost figurate ca având câte o stiva parțială de protocoale pe fiecare dintre interfețele de rețea. Stiva interfeței de intrare va fi folosită pentru a lua decizia de comutare a pachetului (alegerea interfeței de ieşire), un echipament de interconectare putând avea mai mult de două interfețe. După determinarea interfeței de ieşire se va folosi stiva acesteia (parametrii specifici acestei interfețe) pentru eventuala reformatare a pachetului.

Pentru exemplul considerat, fluxul de date între stația A şi stația B, va solicita funcții din mai multe protocoale, acestea fiind prezentate în Figura 8-4.

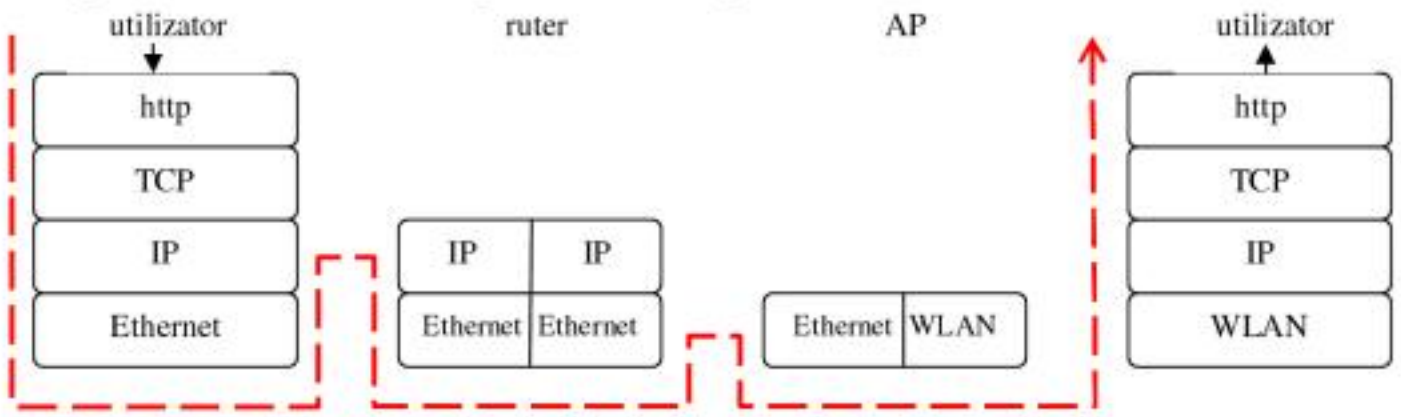


Figura 8-4 Descrierea comunicației între două stații aflate în rețele diferite

8.1.3 Modelul client-server

Pentru implementarea accesului la serviciile de rețea, arhitectura folosită este bazată pe modelul client-server. Acesta se bazează pe definirea a două componente: cea de server şi cea de client.

Un server *este un program (sau prin extindere o stație) ce permite primirea de cereri de la alte entități din rețea, oferind acestora un serviciu.*

Un client *este un program ce oferă mecanismul de interogare a unui server şi eventual de formatare a răspunsului primit de la acesta.*

Un exemplu de model client server este serviciul de web. Acest serviciu presupune rularea unui server HTTP (ex: Apache, Microsoft IIS, etc). Aplicația client poate fi orice navigator (*broswer*).

Chiar pentru exemplul serviciului de web se poate observa că aplicația client integrează în realitate mai multe aplicații client: pentru HTTP, dar şi pentru FTP, TFTP.

Modelul client-server oferă centralizarea serviciilor şi a fost în primul rând promovat ca o metodă de reducere a costurilor totale a unui rețele. Ca urmare a scăderii costului resurselor hardware principală motivație a centralizării serviciilor a devenit, monitorizarea şi securizarea rețelei.

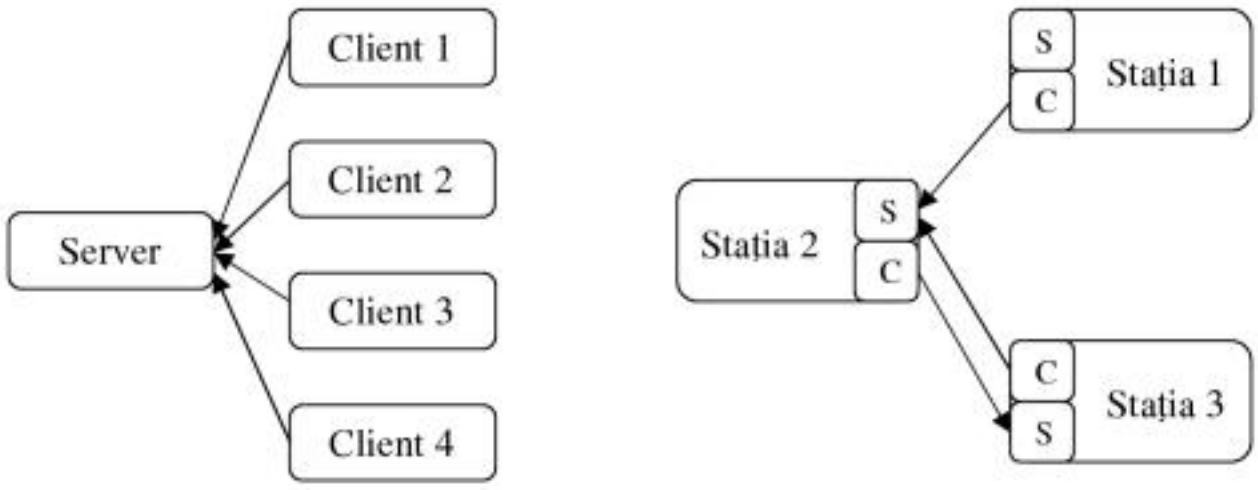


Figura 8-5 Modelul client-server şi modelul peer-to-peer

Noțiunea de server s-a extins şi pentru sistemele de calcul, specificațiile hardware ale unui server depinzând de serviciul pe care îl îndeplineşte. Spre exemplu, un server de stocare va avea un procesor relativ modest, dar un spațiu impresionant de stocare, precum şi mecanisme de redundanță a discurilor.

Modelul punct-la-punct (*peer to peer*) este de fapt un model client-server ascuns. Fiecare entitate ce participă la comunicație poate juca atât rol de client cât şi rol de server.

Spre deosebire de modelul client-server modelul punct-la-punct este un model descentralizat.

Multe din programele de partajare de fişiere utilizează acest model pentru a nu folosi un server care să intermedieze comunicația. Un model client-server de partajare a fişierelor între clienți presupune ca orice pachet schimbat între cei doi clienți să treacă prin server, cea ce aduce atât o latență, dar şi o încărcare a rețelei şi a serverului.

Principalul dezavantaj al centralizării serviciilor este că serverul va fi un punct critic în rețea: în momentul în care serverul nu funcționează, serviciul este întrerupt. Într-o rețea punct-la-punct acest dezavantaj este evitat.

8.1.4 Porturi

În scopul reducerii costurilor unei reţele în general mai multe servicii vor fi oferite de aceeaşi maşină. Spre exemplu, pentru o reţea locală, serverele de DHCP, şi DNS pot fi integrate pe o singură staţie, deoarece resursele solicitate de fiecare serviciu sunt relativ reduse.

Cum va putea totuşi un astfel de server să diferenţieze între cererile de rezolvare de nume (cereri DNS) şi cele de alocare de adrese IP (cereri DHCP)?

Ambele protocoale folosesc la nivelul transport UDP, deci nu vom putea lua decizia pe baza tipul protocolului de nivel transport.

Atât UDP, cât şi TCP folosesc un mecanism de identificare a serviciilor (a protocolului de nivel aplicaţie) denumit port.

***Portul** este un număr reprezentat pe 16 biţi, ce identifică la nivelul transport (în informaţia adăugată de protocoalele TCP sau UDP) un serviciu specific.*

Mecanismul de porturi permite atât oferirea simultană a mai multor servicii, cât şi diferenţierea între cereri de răspuns venite din conexiuni diferite, adică folosirea mai multor servicii simultan. Astfel în antetul de nivel transport vom preciza atât un port destinaţie, ce identifică serviciul apelat, cât şi un port sursă, pe care este aşteptat răspunsul. O conexiune va fi descrisă prin precizarea parametrii de nivel reţea şi nivel transport: adresa IP destinaţie, adresa IP sursă, port destinaţie şi port sursă.

Dacă de pe aceeaşi staţie sunt lansate două sesiuni de telnet către acelaşi server, se creează două conexiuni diferite având aceleaşi adrese IP (sursă şi destinaţie), acelaşi port destinaţie, dar portul sursă diferit.

Porturile sunt numere reprezentate pe 16 biţi, valorile definite pentru porturi fiind cuprinse în intervalul [1, 65535]. Porturile până la 1023 sunt rezervate pentru cele mai importante protocoale ale Internet-ului, fiind denumite porturi rezervate (*known ports*). Porturile rezervate sunt asociate unui protocol pe care serverul îl foloseşte în oferirea unui serviciu.

20	FTP (data)	80	HTTP
21	FTP (conexiune)	110	POP3
22	SSH	143	IMAP
23	telnet	161	SNMP
25	SMTP	443	HTTPS
53	DNS	993	IMAPS

Figura 8-6 Protocoalele importante şi porturile asociate

Rezumând: fiecare serviciu de reţea este oferit folosind protocol de nivel aplicaţie, fiecare protocol are la rândul său asociat un port, în unele cazuri chiar două.

În tabelul de mai sus sunt prezentate câteva dintre protocoalele importante şi porturile folosite. Putem observa spre exemplu că serviciul de FTP are rezervate două porturi: 21 pentru stabilirea conexiunii şi 20 pentru transferul efectiv de fişiere.

Pentru a determina serviciile oferite pe staţia locală va trebui să afişăm lista porturilor deschise (aflate în stare LISTEN). Pentru aceasta putem folosi utilitarul `netsat`: (descrierea utilitarului `netsat` este făcută în 9.3.1.3)

```
kiwi# netstat --tcp --listening
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address     State
tcp        0      0 *:sunrpc                *:*                 LISTEN
tcp        0      0 *:www                   *:*                 LISTEN
tcp        0      0 *:webcache              *:*                 LISTEN
tcp        0      0 *:auth                  *:*                 LISTEN
tcp        0      0 *:ftp                   *:*                 LISTEN
tcp        0      0 *:smtp                  *:*                 LISTEN
```

Din rezultatul comenzii de mai sus se poate deduce că portul ftp (adică portul 21) este deschis. Altfel spus, există o aplicație ce folosește acest port pentru a accepta conexiuni. În general, serviciile sunt rulate pe porturile rezervate, dar prin configurarea serviciilor putem schimba portul pe care se vor accepta conexiunile. Pentru a obține mai multe informații despre aplicația ce rulează pe portul 21 putem folosi clientul de telnet ce va deschide o conexiune pe acest port:

```
kiwi:~$ telnet kiwi.cs.pub.ro 21
Trying 141.85.99.5...
Connected to kiwi.cs.pub.ro.
Escape character is '^]'.
220 Welcome to Kiwi FTP service. vsFTPd 2.0.3 is in service
```

Dacă oprim serviciul de FTP şi reconfigurăm serverul de SSH să primească cereri pe portul 21 rezultatul comenzi telnet va fi:

```
kiwi:~$ telnet kiwi.cs.pub.ro 21
Trying 141.85.99.5...
Connected to kiwi.cs.pub.ro.
Escape character is '^]'.
SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
```

Din motive de securitate serverele pot fi configurate să nu afişeze informații referitoare la versiunea serviciului, sau chiar să afişeze informații false.

În cazul în care se încearcă inițierea unei sesiuni de telnet pe un port ce nu este deschis va fi întors mesajul de conexiune refuzată:

```
kiwi:~$ telnet kiwi.cs.pub.ro 29
Trying 141.85.99.5...
telnet: Unable to connect to remote host: Connection refused
```

8.2 Execuția comenzilor la distanță

Serviciul de execuție al comenzilor la distanță (*remote connection*) este oferit de mai multe protocoale cele mai importante fiind: telnet, rlogin, RSH, SSH.

Protocoalele telnet, rlogin şi RSH se bazează pe realizarea unei conexiuni necriptate (în text clar). Spre deosebire de acestea protocolul SSH stabileşte o conexiune criptată, conexiune folosită atât pentru lansarea comenzilor la distanță, cât şi pentru transferul de fişiere.

8.2.1 Telnet

În practică, graniță dintre serviciu de rețea şi protocol de nivel aplicație este, deseori, una greu de evidențiat. Astfel, putem spune că serviciul de execuție al comenzilor la distanță este oferit de protocolul telnet, dar la fel de corect se poate vorbi de serviciul telnet. Prin serviciul telnet se înțelege posibilitatea execuției comenzilor la distanță peste o conexiune nesigură (necriptată).

Pe lângă denumirea unui protocol de nivel aplicație, și a unui serviciu de rețea, telnet este și denumirea aplicației ce implementează protocolul (a procesului care din punctul de vedere al sistemului de operare oferă serviciul de rețea).

Aplicația telnet este o aplicație client-server: procesul server se numește telenetd, iar clientul se numește telnet. Serverul de telnet este responsabil cu acceptarea sesiunilor inițiate de clienți. O sesiune de telnet pune la dispoziția aplicației client (după autentificare) un interpretor de comenzi ce rulează pe server. În mod implicit, nu există diferențe între resursele disponibile unui utilizator în urma rulării unui interpretor de comenzi local, și resursele accesibile printr-o sesiune de telnet de către același utilizator.

```
[rookie@apple ~]$ telnet 141.85.99.5
Trying 141.85.99.5...
Connected to 141.85.99.5.
Escape character is '^]'.
Fedora release 7 (Moonshine)
Kernel 2.6.21-1.3194.fc7 on an i686
login: manager
Password:
[manager@kiwi ~]$
```

În exemplul de mai sus utilizatorul rookie inițiază o sesiune telnet către serverul 141.85.99.5. După afișarea mesajelor de întâmpinare (denumite MotD – *Message of the Day*) este afișat prompt-ul de login. Este folosit numele de utilizator manager și parola asociată acestui cont. După autentificare va apărea prompt-ul interpretorului de comenzi.

Protocolul telnet a fost standardizat pentru a permite clientului de telnet inițierea de conexiuni la distanță și către alte servere. Majoritatea implementărilor serverelor permit inițierea unei conexiuni prin clientul de telnet (cu unele excepții notabile, precum clienții RDS, cărora le este blocat portul 25 al serviciului SMTP).

Pentru a iniția o conexiune pe serverul de HTTP ce rulează pe stația cu adresa IP 141.85.99.5 trebuie rulată aplicația client telnet cu precizarea destinației și a portului pe care rulează serviciu (implicit 80). După stabilirea sesiunii se pot executa inclusiv comenzi ce respectă sintaxa protocolului HTTP, spre exemplu pentru a obține o pagină se folosește comanda HTTP `GET`, urmată de pagina ce se dorește afișată.

```
kiwi:~$ telnet 141.85.99.5 80
Trying 127.0.0.1...
Connected to localhost.localdomain.
Escape character is '^]'.
GET /~razvan/test.html HTTP/1.0

HTTP/1.1 200 OK
Date: Fri, 03 Nov 2006 19:57:10 GMT
Server: Apache/1.3.33 (Debian GNU/Linux) PHP/4.3.10-16
Last-Modified: Fri, 03 Nov 2006 19:56:40 GMT
ETag: "253095-44-454b9ef8"
Accept-Ranges: bytes
Content-Length: 68
Connection: close
Content-Type: text/html; charset=iso-8859-1

<br />Pagina de test pentru cursul USO - Servicii de Retea <br />
Connection closed by foreign host.
```

Este important de reținut că pentru terminarea unei sesiuni de telnet se poate folosi combinația CRTL+].

8.2.2 SSH

SSH (*Secure Shell*) este un protocol de nivel aplicație, folosit în rețele de calculatoare pentru asigurarea unui canal securizat de transmitere a informației.

Protocolul funcționează după modelul client-server. Pentru a iniția o conexiune, clientul de SSH face o cerere la serverul de SSH, iar acesta din urmă, după autentificarea clientului, permite transferul de date.

Cea mai întalnită implementare a protocolului SSH 2.0 este varianta open-source numită OpenSSH.

Protocolul este adesea folosit pentru executarea comenzilor la distanță, dar poate fi deasemenea utilizat pentru transfer securizat de fişiere sau tunelarea altor protocoale mai nesigure (FTP). Există multe alte protocoale de nivel aplicație, mai vechi, cu care se pot obține aceleaşi efecte precum cu SSH: FTP, rlogin, rsh, telnet. Toate aceste protocoale au însă în comun aceeaşi slăbiciune: metode de autentificare slabe, ce presupun trimiterea parolelor pe rețea în format text clar (*clear text*). Formatul text clar reprezintă un risc maxim de securitate, captarea parolelor făcându-se relativ simplu de o persoană cu acces la rețea.

Pentru asigurarea securității, SSH foloseşte algoritmul RSA de generare de chei publice şi private. După cum denotă şi numele celor două chei, cheia privată trebuie păstrată întotdeauna secretă, iar cea publică poate fi cunoscută de oricine. Există o strânsă relație matematică între cele doua chei, care face imposibilă derivarea cheii private din cea publică, dar care permite decriptarea unui mesaj criptat cu una dintre chei, folosind cealaltă cheie. Dacă se doreşte trimiterea unui mesaj privat către destinatarul X, tot ce trebuie făcut este criptarea mesajului cu cheia publică a destinatarului (fiind publică, toată lumea are acces la această cheie). Odată criptat, acest mesaj poate fi decriptat doar de destinatarul X, căci el este singurul care deține cheia privată corespunzătoare cheii publice cu care s-a făcut criptarea.

8.2.2.1 Utilizarea SSH pentru efectuarea conectării la distanță

Comanda de conectare la un server SSH are doi parametrii importanți: numele utilizatorului şi adresa (numele) serverului destinație. Serverul de SSH la care se va face conectarea: `securessh.pub.ro` (în exemplul de mai jos), acest nume este un nume public, pe care serviciul de DNS îl va traduce într-o adresă IP. Dacă se doreşte conectarea la un server dar nu se cunoaşte numele său DNS, se poate introduce direct IP-ul serverului. Când un client inițiază o conexiune ssh pe un server, trebuie să se conecteze pe un utilizator existent pe acel server pentru a putea avea acces la interpretorul de comenzi. Parametrul `bogdand` specifică utilizatorul în contul căruia se va intra la stabilirea conexiunii.

```
kiwi:/$ ssh bogdand@securessh.pub.ro
Password:
Last login: Wed Sep 19 14:37:29 2007 from 86.121.138.243
Welcome to the dark side.. we've got cookies!
securessh:/$
```

În rezultatul comenzii de mai sus se poate observa faptul ca s-au mai făcut conexiuni anterioare pe acel server de SSH, fiind oferită atât ora şi data ultimei conexiuni, cât şi adresa IP de la care s-a realizat conexiunea.

Când un client de ssh inițiază pentru prima dată o conexiune către un server nou, acesta din urmă va trebui sa se autentifice către client. Fiecare server de SSH are un identificator unic (*host key*) cu care se autentifică clienților. Acest ID este implementat prin chei publice şi chei private. La primirea unei conexiuni de la un client, serverul oferă cheia sa publică, făcând astfel posibilă autentificarea. La acceptarea cheii publice de la server, clientul adaugă această cheie în fişierul `known_hosts`. Astfel, următoarea conexiune pe care acest client o va face la server nu va mai necesita transferul cheii publice, aceasta existând deja stocată pe client.

```
kiwi:/$ ssh bogdand@82.77.5.245
The authenticity of host '82.77.5.245 (82.77.5.245)' can't be established.
RSA key fingerprint is 11:94:89:c3:e5:fe:c3:6c:2e:ab:f6:e2:bd:6d:bb:e7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '82.77.5.245' (RSA) to the list of known hosts.
Password:
Have a lot of fun...
bogdand@server:~>
```

Un utilitar pentru sistemele Windows ce permite conectarea în linie de comandă la un server de SSH este aplicația PuTTY. PuTTY este disponibil gratuit și este un client pentru protocolul SSH, telnet, rlogin, dar poate fi folosit și în comunicațiile *TCP raw*.

O privire de ansamblu asupra opțiunilor oferite de clienții importanți de SSH poate fi găsită la: http://en.wikipedia.org/wiki/Comparison_of_SSH_clients.

8.2.2.2 Utilizarea SSH pentru rularea de comenzi la distanță

Sintaxa completă a utilitarul `ssh` permite precizarea unei liste de parametri a utilizatorului și a adresei destinație, precum și a unei comenzi ce va fi rulată după stabilirea sesiunii SSH. Dacă nu este precizată o comandă se va rula un interpretor de comenzi (cel mai adesea `/bin/bash`).

În exemplul de mai jos, sunt rulate local două comenzi separate prin *";"*. Apoi se rulează aceleași comenzi (de data aceasta protejate între ghilimele), rezultatul afișat este cel al executării lor pe stația destinație, după autentificarea cu utilizatorul bogdand. Se observă din exemplu că autentificarea se realizează fără a interoga utilizatorul, aceasta fiind rezultatul unei autentificări pe bază de chei (vezi 8.2.2.5).

```
rrazvan@apple:~$ hostname; pwd
apple
/home/rrazvan
rrazvan@apple:~$ ssh bogdand@141.85.99.5 "hostname;  pwd"
kiwi
/home/users/bogdand
rrazvan@apple:~$
```

8.2.2.3 Copierea fișierelor la distanță

Efectuarea transferului de date se poate face prin utilitarul `scp`, ce face parte din pachetul `ssh`. Ideea din spatele `scp` este de a folosi canalul securizat oferit de `ssh`, pentru a face transfer de date în format text sau binar între client și server. Sintaxa `scp` este asemănătoare comenzii tradiționale `cp`:

```
scp sursa destinație
```

Dacă se dorește transferul a mai multe fișiere sau directoare, destinația trebuie să fie un director existent. Pentru a efectua o copiere de fișier de pe client, în directorul `/home/bogdand/work/` de pe serverul cu adresa IP 82.77.5.245, comanda este:

```
kiwi:~/$ scp CiscoTFTP.exe bogdand@82.77.5.245:~/work
Password:
CiscoTFTP.exe                                 100% 1296KB 129.6KB/s   00:10
kiwi:~/$
```

Pentru platforma Windows o aplicație *open source* ce poate fi folosită pentru partajarea de fișiere peste o conexiune SSH este WinSCP. Aceasta oferă o interfață grafică pentru transferul de fișiere prin SCP, SFTP sau FTP.

8.2.2.4 Serverul de SSH

Pentru a putea permite conexiuni SSH spre stația locală, trebuie instalat OpenSSH server. Fișierul de configurare pentru serverul de SSH, se află în `/etc/ssh/sshd_config`, și conține directive standard precum:

- portul pe care se accepta conexiuni SSH
- opțiunea de a putea rula aplicații X peste conexiunea criptată de SSH

```
kiwi:/etc/ssh$ vim sshd_config
[...]
 Port 22
 X11Forwarding yes
[...]
```

8.2.2.5 Autentificarea cu chei

Pentru o securitate sporită, cu SSH, se poate folosi autentificarea cu chei publice şi chei private. În cazul standard al folosirii SSH, fiecare stație păstrează o asociere între utilizator şi parolă în mod automat. Dacă se doreşte însă utilizarea criptării bazată pe chei, devine obligația utilizatorului de a crea această asociere manual. Pentru a genera o pereche de chei se poate folosi utilitarul `ssh-keygen` instalat odată cu pachetul OpenSSH.

```
kiwi:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/waters/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/waters/.ssh/id_rsa.
Your public key has been saved in /home/waters/.ssh/id_rsa.pub.
The key fingerprint is:
c1:45:40:92:b6:f6:e5:0b:72:23:73:ff:88:58:27:d0 waters@waters-laptop
kiwi:~$
```

Odată generate, cele două chei sunt stocate în doua fişiere: `id_rsa` (cheia privată) şi `id_rsa.pub` (cheia publică).

În momentul creării cheilor, utilizatorului i se va cere un şir de caractere (*passphrase*) care identifică perechea de chei, şi care va fi cerut ulterior, în locul parolei. Acest şir poate fi vid, caz în care nu se va mai cere intervenția utilizatorului în procesul de autentificare. Folosirea unui *passphrase* vid devine deosebit de util pentru automatizarea unor acțiuni (prin scripturi).

Pentru a copia cheia publică pe stația aflată la distanță, se poate folosi în mod asemănător utilitarul `ssh-copy-id`, specificând ca parametru, adresa serverului SSH.

```
kiwi:/# ssh-copy-id -i /home/waters/.ssh/id_rsa 86.121.138.243
```

La următoarea încercare de conectare prin SSH, singura diferență va fi cerea *passphrase*-ul în locul unei parole de cont, însă din punct de vedere al securității, conexiunea va fi mult mai sigură.

8.2.3 Utilitarul wget

`wget` este un utilitar gratuit (GNU Project), creat pentru a facilita o descarcare simplă şi eficientă a resurselor de pe WEB. Aceste resurse pot fi obținute cu `wget` prin două dintre cele mai populare protocoale folosite în Internet: HTTP si FTP. Numele utilitarului este obținut de fapt printr-o alăturare semantică a World Wide Web (W) şi a metodei `HTTP GET`.

Principalul avantaj al `wget` este faptul că execută o **descărcare neinteractivă** a fişierelor. Într-o aplicație din mediul grafic (Firefox, Filezilla), o descărcarea uzuală necesită prezența pe sistem a utilizatorului. Cu alte cuvinte, acesta trebuie să fie logat pentru a se putea efectua operația de descărcare, sau pentru a avea posibilitatea de a reactiva o descărcarea întreruptă de către server.

În contrast, folosind `wget`, utilizatorul poate să se autentifice pe sistem, să pornească descărcarea, şi să se deautentifice, `wget` continuând să ruleze pe sistem până la terminarea

sarcinilor neîncheiate. Mai mult, dacă conexiunea către server este întreruptă, utilitarul va stoca fişierele luate până în acel moment pe hard-disc, va salva parametrii sesiunii şi adresa serverului şi va continua sarcina de descărcare în momentul în care legătura cu serverul se reface şi resursa este din nou accesibilă.

Prezentând un astfel de avantaj evident faţa de clienţii de descărcare din mediul grafic, `wget` a fost rapid adoptat de marea majoritate a utilizatorilor care aveau o conexiune înceată la Internet şi care nu puteau petrece timpul necesar unei descărcări interactive, autentificaţi pe sistem. Pe măsură ce WEB-ul a evoluat şi utilitarul preferat de descărcare al utilizatorului ce lucrează în linie de comandă a adăugat noi funcţionalităţi.

Un avantaj notabil al utilitarului `wget` este capabilitatea de descărcare si de păstrare pe client a structurii de directoare de pe server, fiind de asemenea posibilă indicarea nivelului de adâncime până la care să fie descărcată structura de directoare. Acest tip de descărcare, numit descărcare recursivă, a fost extins până la posibilitatea de a descărca pe client, site-uri întregi cu toate paginile si legăturile dintre acestea. Considerând situaţia unei reţele locale, descărcarea unui site pe maşina server din reţea (proces numit si oglindirea site-ului), ar face implicit posibilă accesarea acelui site în mod *offline* de către toţi clienţii din reţeaua locală.

8.2.3.1 Modul de apelare al utilitarului în linia de comandă

Pentru a descărca un fişier prin HTTP se poate folosi comanda:

```
kiwi:$ wget -c http://ccnp.catc.ro/resources/CiscoTFTP.exe &

--15:57:06--  http://ccnp.catc.ro/resources/CiscoTFTP.exe
           => `CiscoTFTP.exe'

Resolving ccnp.catc.ro... 141.85.37.2
Connecting to ccnp.catc.ro|141.85.37.2|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1,327,497 (1.3M) [application/x-msdos-program]

100%[====================================>] 1,327,497      2.66M/s

15:57:07 (2.66 MB/s) - `CiscoTFTP.exe' saved [1327497/1327497]

kiwi:$
```

Opţiunea "`-c`" asigură continuarea descărcării în cazul întreruperii conexiunii cu serverul, iar simbolul "`&`" specifică rularea utilitarului în background .

Descărcarea unui site pe discul local pentru a face posibilă accesarea sa *offline:*

```
kiwi:$ wget --convert-links -r http://www.gnu.org/ -o gnulog
kiwi:$ ls
gnulog  www.fsf.org  www.gnu.org
```

Parametrul "`--convert-links`" transformă legăturile (linkurile) din site pentru ca site-ul să poată fi vizionat *offline*.

Descărcarea tuturor legăturilor de pe siteul www.yahoo.com cu o adâncime de 1.

```
kiwi:$ wget -r -l1 www.yahoo.com
```

8.2.4 Utilitarul cURL

Utilitarul cURL este aplicaţie pentru transferul de fişiere ce îşi propune să ofere suport atât pentru cerinţe avansate de securitate, cât şi pentru un număr foarte mare de protocoale. Lista protocoalelor implementate de cURL include FTP, HTTP, telnet, LDAP, precum şi protocoale orientate spre securitate precum SCP, SFTP, FTPS, HTTPS. Pentru asigurarea securităţi se pot folosi în cURL numeroase metode de autentificare pe bază de parolă, cât şi pe certificate.

Utilitarul cURL ce se execută din linia de comandă, ceea ce permite includerea sa în scripturi. Cel mai adesea cURL este folosit pentru scripting HTTP, în scopul simulării activității utilizatorilor ce folosesc diferite broswere.

Utilitarul cURL implementează numeroase mecanisme specifice clienților web. Spre exemplu oferă gestionarea cookies (trimiterea şi salvarea locală), navigarea prin pagini de formular, folosirea unui proxy web, urmărirea redirectărilor automate, sau chiar schimbarea semnăturii navigatorului.

Spre exemplu, presupunem că în cadrul unui site se află un formular la adresa: `test.cs.pub.ro/login.cgi`. În acest formular trebuie completat un câmp de nume şi apoi apăsat butonul OK pentru a accesa site-ul. Formularul este scris folosind sintaxa HTML:

```
<form method="GET" action="login.cgi">
    <input type=text name="nume">
    <input type=submit name=press value="OK">
</form>
```

Cererea poate fi trimisă folosind cURL astfel:

```
curl "test.cs.pub.ro/login.cgi?nume=Razvan&press=OK"
```

În cazul unui formular ce permite clientului să încarce un fişier se poate folosi cURL cu opțiunea `"-F"`. Presupunem că formularul este scris astfel:

```
<form method="POST" enctype='multipart/form-data' action="upload.cgi">
    <input type=file name=upload>
    <input type=submit name=press value="OK">
</form>
```

Pentru formularul aflat la adresa `test.cs.pub.ro/upload.cgi` va fi încărcat fişierul test01 şi apoi simulată apăsarea butonului OK:

```
curl -F upload=@test01 -F press=OK test.cs.pub.ro/upload.cgi
```

Utilitarul cURL oferă suport şi pentru schimbarea identității navigatorului. Pentru a simula o cerere către serverul test.com inițiată de un navigator Internet Explorer 7.0 ce rulează pe un sistem Windows 2000 vom folosi comanda:

```
curl -A "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.0)" test.co
```

8.3 Email - Poşta electronică

Poşta electronică (electronic mail), *abreviat e-mail sau email, este o metodă de compunere, transmitere şi recepție a mesajelor peste sisteme de comunicație electronică.*

Ideea de mesagerie electronică datează din anul 1971, când Ray Tomlinson dezvolta prima aplicație de email pentru ARPANET. Aceasta era formată din două programe: SNDMSG, utilizat pentru a transmite mesaje, respectiv READMAIL, folosit pentru citirea mesajelor. În anul 1972, comenzile MAIL şi MLFL au fost adăugate programului FTP. Aceasta a fost modalitatea de transmitere a mesajelor în rețeaua ARPANET până la începutul anilor '80, când a fost dezvoltat protocolul SMTP. În scurt timp, serviciul de email a crescut popularitatea ARPANET-ului şi a devenit aplicația principală a acestuia.

Serviciul de email a oferit la început doar transferul de mesaje text între utilizatori aflați în rețele diferite. Ulterior serviciul a fost extins şi la transmiterea de fişiere binare. Standardul de codificare a informației binare se numeşte MIME (*Multipurpose Internet Mail Extensions*).

8.3.1 Arhitectură şi funcţionare

Pentru livrarea mesajelor se foloseşte un mecanism de adresare ce apelează atât ierarhia de nume din Internet (serviciul de DNS) pentru a preciza serverul, cât şi informaţii relevante local, precum numele utilizatorului.

O adresă de email conţine două tipuri de informaţii separate prin simbolul `"@"` (citit la sau *at* sau *a-round*):

- numele utilizatorului care doreşte transmiterea sau recepţia mesajului;
- numele (DNS) al server-ului care transmite/recepţionează mesajul.

Pe baza adresei de email destinaţie serviciul de poştă electronică livrează mesajele de la clienţii de email, folosind servere de email, către căsuţa poştală asociată cu respectiva adresă de email.

> ***Căsuţă poştală*** *este o intrare în sistemul local de fişiere utilizată pentru stocarea mesajelor de poştă electronică corespunzătoare unui utilizator.*

Componenta esenţială a serviciului de poştă electronică o reprezintă protocolul **SMTP** (*Simple Mail Transfer Protocol*). Acest protocol oferă mecanisme pentru preluarea mesajelor de la clienţi de email şi livrarea lor pe serverele de email destinaţie.

Odată cu dezvoltarea sistemelor de operare pentru calculatoarele personale, a apărut şi nevoia preluării noilor mesaje pe staţia utilizatorului. Acest lucru este realizat printr-un nou serviciu ce va asigura ridicarea email-ului, sincronizarea căsuţei poştale de pe server cu cea de pe staţia de lucru.

Pentru ridicarea mesajelor de pe serverul de email cele mai populare protocoale sunt **IMAP** (*Internet Message Access Protocol*) şi **POP3** (*Post Office Protocol version 3*). Pentru oferirea acestui serviciu va trebui pornit un daemon de IMAP sau POP3, în plus faţa de serverul de SMTP.

Marea majoritate a clienţilor de email integrează pe lângă componenta de client SMTP (folosită pentru transmiterea mesajelor) şi componente de client IMAP şi POP3 pentru interogarea căsuţei poştale de pe serverul de email.

O deosebire importantă faţă de POP3, o reprezintă extinderea posibilităţilor de prelucrare a mesajelor din căsuţa poştală oferite de protocolul IMAP. Folosind IMAP mesajele nu sunt descărcate de pe server decât la vizualizarea lor, existând şi opţiunea descărcării doar a unei părţi din mesaj.

Rezumând, arhitectura serviciului de email include două componente principale: un serviciu pentru primirea şi trimiterea mesajelor şi un serviciu pentru ridicarea mesajelor de pe serverul de email.

Spre deosebire de POP3, unde toate mesajele se găsesc pe server într-o singură căsuţă poştală (*inbox*), iar directoarele nu se pot crea decât în aplicaţia client, protocolul IMAP permite crearea de directoare direct pe server. Utilizând comenzi IMAP, aplicaţia client poate folosi filtre pentru mutarea mesajelor dintr-un director în altul, fără să fie nevoie măcar ca *mailbox*-urile să fie situate pe acelaşi server.

8.3.1.1 Trimiterea unui mesaj

Pentru a exemplifica modul în care componentele prezente mai sus interacţionează, vom considera următoarea situaţie: Dan are adresa de mail dan@a.org şi doreşte să-i transmită un mesaj lui Bogdan care are adresa de mail bogdan@b.org.

Pentru livrarea unui mesaj în căsuţa poştală asociată contului bogdan@b.org paşii sunt prezentaţi în Figura 8-7.

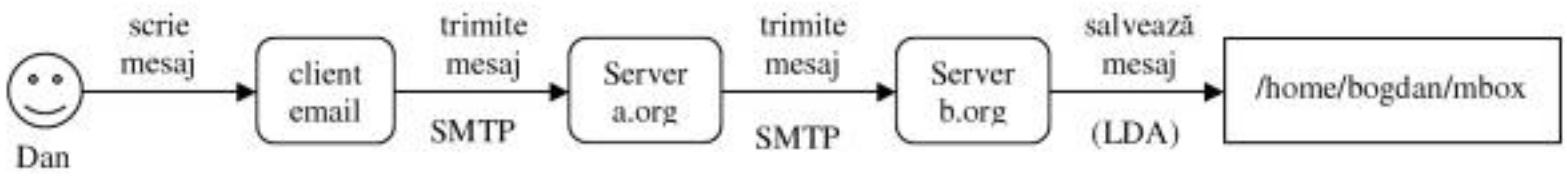


Figura 8-7 Livrarea unui mesaj

Dan va deschide un client de email în care va edita mesajul. Când Dan va da comanda de trimitere a mesajului, clientul de email va iniția o conexiune SMTP către serverul de mail local. Pentru stabilirea conexiunii, serverul va cere autentificarea utilizatorului Dan. În general, clientul de mail va salva numele de utilizator şi parola şi va realiza autentificarea conexiunii SMTP în mod transparent pentru utilizator. După autentificare clientul de email va formata mesajul conform protocolului SMTP şi îl va trimite către server.

Serverul de mail local va analiza apoi adresa de email destinație, pe baza căreia va determina domeniul de nume în care trebuie livrat mesajul (în cazul exemplului: b.org). Va iniția o cere de rezolvare de nume (cerere DNS) pentru a determina adresa serverului responsabil cu preluarea mailului pentru domeniul b.org (acest server se numeşte *mail exchanger*). După determinarea serverului de mail pentru b.org, serverul local va trimite mesajul folosind protocolul SMTP.

Serverul b.org va stoca mesajul în căsuța poştală a utilizatorului bogdan. La nivelul acestui pas se poate interpune o aplicație de distribuție a mesajelor denumită generic LDA (*Local Delivery Agent*). O astfel de aplicație poate realiza filtrarea mesajelor, livrarea anumitor mesaje în alte directoare, oprirea spamurilor, etc. În Linux cea mai folosită aplicație de distribuție a mesajelor este oferită de pachetul procmail.

8.3.1.2 Ridicarea mesajelor

Pentru a ridica mesajul din căsuța poştală vor fi urmați paşii din Figura 8-8:

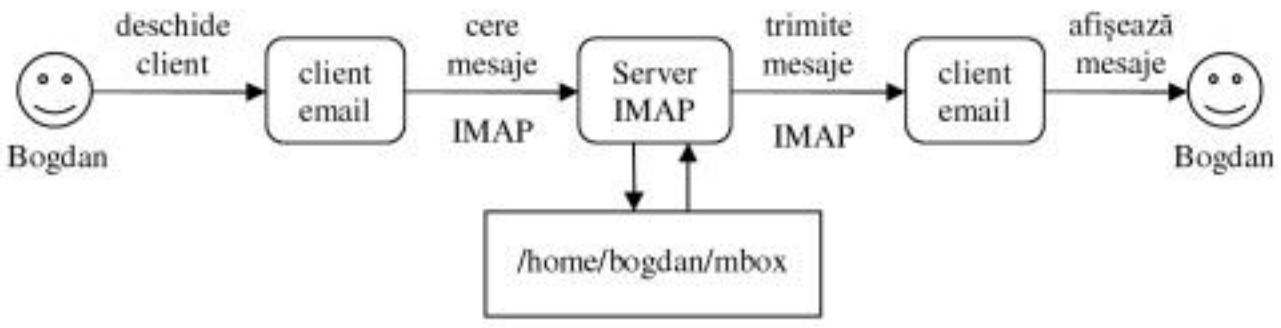


Figura 8-8 Ridicarea mesajelor de pe server

Bogdan va deschide clientul de mail, iar acesta va iniția o sesiune de IMAP către serverul de mail. După autentificare serverul va deschide căsuța poştală a utilizatorului Bogdan. Mesajele ridicate din căsuța poştală în sesiunile anterioare de IMAP au fost marcate, astfel serverul va selecta doar noile mesaje (ce nu sunt marcate) şi le va transmite prin sesiunea de IMAP către clientul de email. Înainte de a încheia sesiunea, clientul de email va informa serverul asupra listei mesajelor ce au fost şterse din căsuța poştală aflată pe stația utilizatorului, permițând astfel serverului să sincronizeze conținutul căsuței poştale locale cu cea aflată pe stație.

După ridicarea mesajelor de pe server, căsuța poştală poate fi consultată inclusiv *offline*, folosind clientul de email.

8.3.1.3 Folosirea webmail pentru citirea şi transmiterea mesajelor

O altă metodă disponibilă utilizatorilor pentru citirea şi trimiterea mesajelor este folosirea unui server de web pentru interfaţa cu serverul de SMTP, serviciu rezultat fiind denumit **webmail**.

Folosind exemplul anterior, mesajul livrat prin SMTP a fost stocat în căsuţa poştală a utilizatorului Bogdan. Utilizatorul va lansa un client web (un broswer) şi va iniţia o conexiune HTTP către serverul de web ce rulează pe acelaşi sistem cu serverul de SMTP responsabil cu primirea mesajelor pentru domeniul b.org. Serverul de web va oferi o interfaţă ce permite operaţii direct pe căsuţa poştală aflată pe server: ştergerea/mutarea mesajelor, marcarea lor, crearea de directoare, etc.

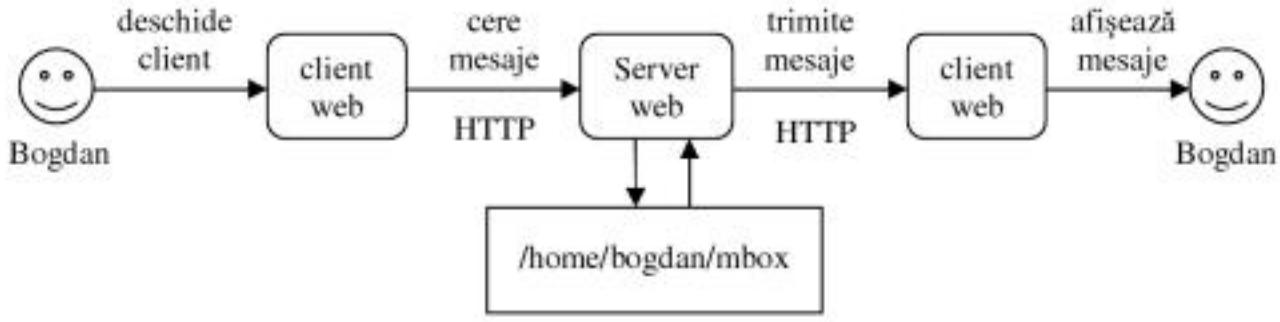


Figura 8-9 Citirea mesajelor folosind webmail

8.3.2 Servere de email

După cum a fost prezentat în subcapitolul anterior, serviciul de email cuprinde în principal două componente: un server de SMTP pentru trimiterea şi primirea de mesaje, şi un serviciu de ridicare mesajelor, implementat printr-un server ce integrează atât protocolul POP3, cât şi IMAP. În figura de mai jos sunt prezentate componentele unui server de email şi interacţiunea acestora cu o aplicaţie client.

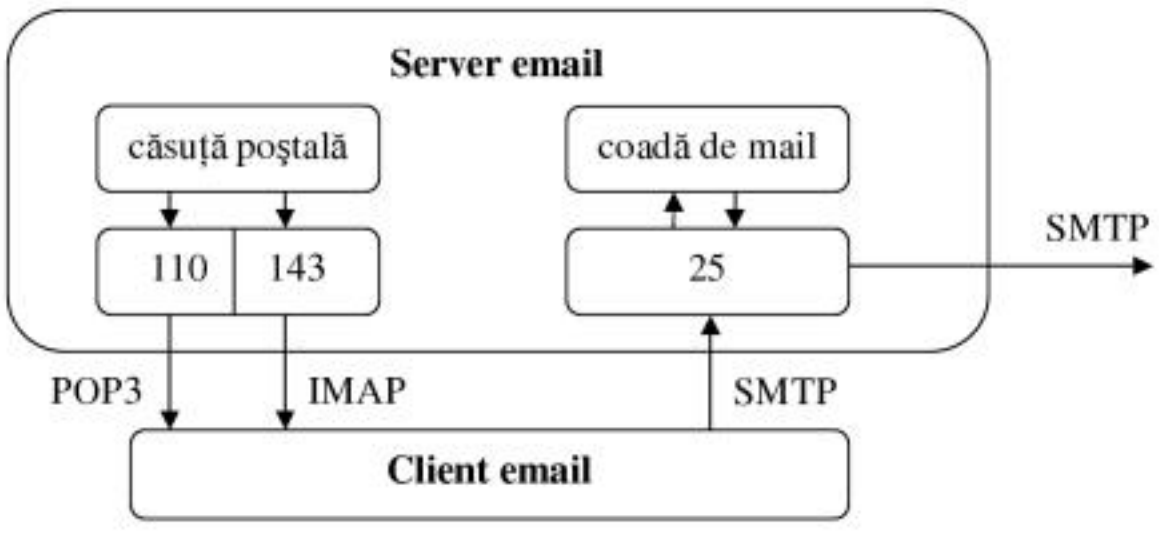


Figura 8-10 Componentele unui server de email

În cazul în care clientul de email rulează pe acelaşi sistem cu serverul SMTP, interogarea căsuţei poştale se poate realiza direct, eliminându-se astfel necesitatea serverului de POP3/IMAP.

Soluţia Microsoft pentru serviciul de email integrează toate componentele mai sus prezentate şi se numeşte Microsoft Exchange Server.

8.3.2.1 Server SMTP

Serverul de SMTP este responsabil cu trimiterea şi recepţionarea mesajelor prin stabilirea unei conexiuni SMTP. Portul folosit implicit pentru comunicaţia SMTP este 25.

Deşi există numeroase aplicaţii pentru serverele SMTP, primele patru aplicaţii ocupă 85% din piaţă conform statisticilor publicate de Security Space:

http://www.securityspace.com/s_survey/data/man.200708/mxsurvey.html

Sendmail a fost scris în anii '80 de Eric Allman şi, timp de mulţi ani, a deţinut supremaţia absolută, fiind singurul server de mail utilizat. Principalul său avantaj este maturitatea, Sendmail fiind foarte bine documentat.

Un dezavantaj al Sendmail este complexitatea fişierului de configurare, deşi există utilitare pentru generarea acestuia. De asemenea, numeroasele probleme de securitate întâlnite de-a lungul timpului constituie un alt punct vulnerabil.

Postfix a apărut ca o alternativă mai rapidă şi mai sigură pentru Sendmail, propunându-şi a fi suficient de asemănător cu aceasta pentru a fi familiar utilizatorilor, dar mult mai uşor de administrat. Postfix a câştigat un număr foarte mare de utilizatori în ultima vreme; deşi Sendmail continuă să fie cel mai utilizat server de email, Exim şi Postfix au câştigat suficient de mulţi utilizatori pentru a ameninţa poziţia de lider deţinută de Sendmail.

8.3.2.2 Coada de email

Mesajele trimise de clienţi, ajunse pe serverul SMTP, sunt adăugate mai întâi într-o coadă de aşteptare. De aici vor fi preluate în ordinea sosirii şi se va încerca livrarea lor la destinaţie. Dacă conexiunea cu serverul SMTP destinaţie nu poate fi stabilită, sau este întreruptă înainte de trimiterea completă a mesajului, respectivul mesaj este trecut la sfârşitul cozii.

Pentru inspectarea cozii de mail se poate folosi în cazul unui server postfix utilitarul `mailq`. În sendmail comanda echivalentă este `sendmail -bp`.

În exemplu de mai jos în coada de mail se află un singur mesaj, ce nu a putut fi livrat deoarece nu a putut fi stabilită o conexiune SMTP cu serverul destinaţie (în cazul acesta Mail-Exchange.ubbcluj.ro)

```
kiwi:~# mailq
-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
CDCBE70156     2162 Thu Sep 13 06:15:06  eczkwopqjfbg@rrkassociates.com
  (connect to Mail-Exchange.ubbcluj.ro[193.230.247.198]: Connection timed out)
                                         john@ubbcluj.ro

-- 3 Kbytes in 1 Request.
```

8.3.2.3 Server POP3/IMAP

POP3 a fost proiectat pentru a permite utilizatorilor cu conexiuni intermitente (cum sunt conexiunile *dial-up*) să extragă mesajele de pe server atunci când sunt conectaţi şi apoi să le vizualizeze şi să le manipuleze fără necesitatea unei conexiuni. Deşi marea parte a clienţilor de mail au opţiunea de a păstra mesajele pe server, în general clienţii de email care folosesc POP3 se conectează, ridică mesajele, le stochează pe calculatorul utilizatorului ca mesaje noi, le şterg de pe server şi se deconectează. Citirea mesajelor descărcate prin POP3 se poate face doar după ce clientul s-a deconectat. În contrast, protocolul IMAP de ridicare a mesajelor electronice suportă atât mod de operare conectat cât şi deconectat.

Imediat după iniţierea conexiunii, se va realiza autentificarea clientului la server. Ca şi alte protocoale mai vechi din Internet, POP3 folosea la început mecanisme de autentificare fără criptare. Deşi încă se practică transmiterea de parole în clar, POP3 suportă câteva metode de autentificare pentru a furniza diverse niveluri de protecţie împotriva accesului nedorit la mesajele utilizatorului. O astfel de metodă este comanda APOP care foloseşte MD5 pentru obţinerea unei chei de autentificare. POP3 suportă de asemenea metode de autentificare de tip IMAP folosind extensia AUTH. Este, de asemenea, posibil să se cripteze trafic POP3 folosind TSL/SSL.

Protocolul POP3 este utilizat în special pentru transferarea mesajelor de pe server pe orice alt calculator pentru a le citi *offline*. În cazul în care se doreşte accesarea unui singur cont de poştă electronică de la mai multe locaţii (acasă/birou), protocolul POP3 este limitat. Acest

dezavantaj a dus la dezvoltarea unui alt protocol, IMAP (*Internet Media Access Protocol*), specificat în RFC 2060. În mod asemănător cu POP3, IMAP permite transferarea mesajelor către un alt calculator. Atunci când IMAP este utilizat în modul *online* clientul nu transferă mesajele şi nici nu le şterge de pe server. Clientul poate cere însă să primească antetele mesajelor, anumite părţi din mesaje, sau chiar mesajele care se potrivesc unui criteriu de căutare. În esenţă, IMAP permite manipularea mesajelor dintr-o căsuţă poştală aflată pe un server ca şi când acestea ar fi stocate local.

Cele mai populare server-e de IMAP sunt Courier IMAP Server, University of Washington imapd şi Cyrus IMAP Server de la Carnegie Mellon University.

8.3.2.4 Căsuţa poştală

Există două implementări diferite pentru stocarea mesajelor: *mailbox* şi *maildir*. În primul caz toate mesajele sunt stocate într-un singur fişier, astfel noile mesaje vor fi adăugate la sfârşitul acestui fişier în ordinea sosirii lor. Pentru *maildir*, fiecare mesaj este păstrat într-un fişier separat, eliminându-se astfel toate problemele cauzate de operaţii concurente pe aceeaşi căsuţă poştală.

În majoritatea distribuţiilor UNIX mesajele noi sunt păstrate centralizat într-un singur director: `/var/mail` (sau pe unele sisteme `/var/spool/mail`). La iniţierea unei conexiuni folosind un client de email (fie pe POP3, IMAP, sau chiar folosind webmail) noile mesaje sunt mutate din fişierul corespunzător utilizatorului în căsuţa poştală a acestuia.

În exemplul de mai jos, după sosirea unui nou mesaj, fişierul de noi mesaje are o dimensiune diferită de zero, cea ce înseamnă că sunt mesaje noi neridicate. După ridicarea mailului se poate observa că fişierul `/var/mai/rrazvan` are dimensiunea zero, iar căsuţa poştală a utilizatorului rrazvan s-a mărit.

```
rrazvan@kiwi:~$
You have new mail in /var/mail/rrazvan
rrazvan@kiwi:~$ ls -l mbox
-rwxr-xr-x 1 rrazvan mail 556 2007-09-15 20:03 mbox
rrazvan@kiwi:~$ ls -l /var/mail/rrazvan
-rw-rw---- 1 rrazvan mail 972 2007-09-15 20:05 /var/mail/rrazvan

rrazvan@kiwi:~$ ls -l mbox
-rwxr-xr-x 1 rrazvan mail 1592 2007-09-15 20:05 mbox
rrazvan@kiwi:~$ ls -l /var/mail/rrazvan
-rw-rw---- 1 rrazvan mail 0 2007-09-15 20:05 /var/mail/rrazvan
```

În figura de jos sunt prezentaţi paşii de la primirea unui mesaj, până la ridicarea acestuia de către clientul de email. La primul pas se iniţiază o conexiune SMTP şi noul mesaj este adăugat la sfârşitul fişierului `/var/mail`. Când clientul de email interoghează serverul de mail (după ce este autentificat), conţinutul /var/mail este mutat în căsuţa poştală a utilizatorului rr. În acest exemplu căsuţa poştală este implementată folosind un director de mail.

În ultimul pas serverul de email livrează noile mesaje clientului de mail folosind o conexiune POP3 sau IMAP.

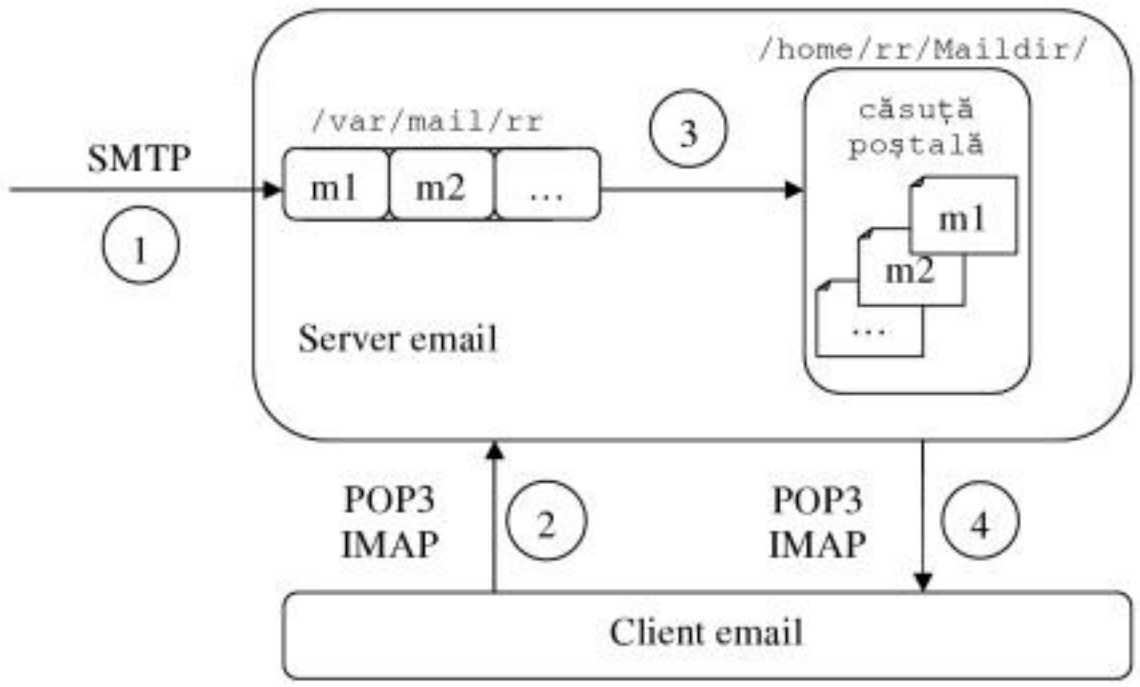


Figura 8-11 Primirea şi ridicarea mesajelor

8.3.3 Clienţi de email

Aşa cum s-a prezentat mai sus, clienţii de email sunt utilitare care permit descărcarea de mesaje de pe un server de email şi stocarea lor local. În mod real, există două situaţii în care este necesară folosirea lor: atunci când contul de email nu poate fi vizualizat printr-o interfaţă de tip web, sau atunci când conexiunea la Internet este slabă sau instabilă, şi este de preferat salvarea corespondenţei pe computerul personal pentru a fi uşor accesată şi ulterior. Există mai mulţi clienţi de email, cei mai importanţi fiind Microsoft Outlook (varianta inclusă în distribuţia de Windows se numeşte Outlook Express) şi Mozilla Thunderbird.

Un avantaj al clienţilor de email este stocarea centralizată a mesajelor primite pe conturi de email diferite. În cele ce urmează, ne vom concentra asupra Mozilla Thunderbird, deoarece este gratuit şi disponibil pentru toate sistemele majore de operare (Windows, Linux, Mac OS).

Configurarea unui client de email presupune în primul rând crearea unui cont pentru fiecare adresă de email ce se doreşte administrata cu Thunderbird. Sunt folosite două protocoale diferite: SMTP pentru a trimite mesaje şi POP3 pentru a primi mesaje. Setarea unui cont presupune introducerea adresei de email, parolei asociate, adresa serverului de SMTP şi a serverului de POP3. Se poate opta pentru salvarea parolei într-un fişier de configurare, sau pentru introducerea ei manuală de fiecare data când se va stabili o conexiune la unul dintre serverele de mail. În cazul în care doar unul dintre cele două servere este setat corespunzător, se vor putea doar descărca (în cazul POP3) sau trimite mesaje (respectiv SMTP).

Unul dintre avantajele Thunderbird este integrarea cu serviciul de email oferit de Google, numit Gmail. Pentru setarea unui cont de Gmail, trebuie parcurşi următorii paşi: se activează pe site-ul Gmail opţiunea de a folosi un server de POP3, apoi din Thunderbird se creează un nou cont, se alege "*Gmail account*" şi se introduce adresa de email. Restul setărilor se efectuează automat.

Pentru a trimite/primi corespondenţă, se foloseşte butonul "Get Mail", comandă ce sincronizează baza de date locală cu cea de pe server. În consecinţă, un email care este scris poate fi trimis instantaneu sau poate fi stocat local şi trimis mai târziu, când va exista conectivitate la Internet. Thunderbird se sincronizează în mod automat cu serverul la intervale regulate de timp.

Thunderbird oferă o gamă de servicii standard: memorarea adreselor de email, sortarea emailurilor după, data primirii, subiect, numele expeditorului, importanţă (crescător sau descrescător). Emailurile pot fi grupate pe threaduri (toate mesajele care fac parte din aceeaşi

discuție), facilitate deosebit de folositoare în cazul mesajelor primite de la o listă de discuții. De asemenea pentru localizarea unui mesaj clientul oferă un motor de căutare ce poate folosii criterii ce vor analiza: adresa sursă, adresa destinație, subiectul mesajului sau conținutul său.

Pentru o mai buna organizare a corespondenței, se poate crea o structură de directoare şi gruparea mesajelor în mod logic, în opoziție cu stocarea tradițională a tuturor email-urilor în căsuța poştală (*inbox*). Aplicarea de filtre permite selectarea emailurilor pe bază de criterii avansate (ce includ adresă, nume expeditor, subiect, conținut) şi efectuarea de acțiuni asupra lor (mutarea in diferite directoare, ştergerea lor, marcarea lor), facilitate extrem de utilă în cazul unui număr mare de mesaje zilnice.

O limitare în folosirea unui soluții mixte bazate pe Thunderbird şi Gmail o reprezintă nesincronizarea marcării mesajelor citite între cele două aplicații.

8.3.4 Securitatea serviciului de email

Cea mai întâlnită problemă asociată cu poşta electronică o reprezintă primirea de **spam** (mesaje nesolicitate). Din păcate, traficul de spam a depăşit 90% din traficul SMTP în Internet.

Pentru a reduce traficul de spam în primul rând trebui ca serverul de SMTP să nu accepte să trimită mesaje de la orice client SMTP din Internet. Majoritatea destinațiilor de mail vor verifica pentru fiecare mesaj dacă nu a fost trimis folosind un server SMTP de tip *open mail relay* (server ce acceptă să trimită mesaje pentru orice client), caz în care vor considera respectivul mesaj drept spam.

Traficul de spam poate fi controlat şi pe serverul destinație înainte de a fi preluat de aplicația client. Reducerea traficului de spam pe serverul de SMTP poate fi făcută prin implementarea a trei tipuri de liste:

- Liste albe – mesajele trimise de un server conținut într-o astfel de listă sunt considerate automat mesaje legitime
- Liste gri – mesajele primite de la servere necunoscute sunt, pentru o perioadă de timp, respinse. Dacă serverul sursă încercă retransmiterea mesajului şi după expirarea acestei perioade, mesajul va fi considerat legitim. Această metodă se bazează pe faptul că serverele de spam nu vor încerca retransmiterea mesajelor.
- Liste negre – mesajele primite de la un astfel de server sunt etichetate ca spam

O altă clasă de aplicații de filtrare a traficului de spam sunt integrate în LDA (*Local Delivery Agent*). Aceste aplicații în general folosesc un set extins de criterii pentru a calcula probabilitatea ca un mail să fie spam.

Filtrarea traficului de spam poate fi realizată inclusiv de clientul de mail. Cu toate acestea, chiar şi în cazul folosirii simultane a tuturor metodelor de control a traficului nedorit mesajele de spam nu pot fi total eliminate.

O altă problemă importantă a serviciului de email este **absența unui mecanism de autentificare a expeditorului**. Există extensii ale protocolului SMTP ce rezolvă această problema, dar care nu se bucură de o popularitate însemnată. Singura soluție cu adevărat eficientă în prevenirea atacurilor bazate pe asumarea unui false identități (numite atacuri *phising*) este instruirea utilizatorilor asupra limitărilor protocolului SMTP.

8.4 WWW

World Wide Web sau WWW este, probabil, cel mai cunoscut şi utilizat serviciu din Internet. WWW-ul se prezintă ca un set de resurse interconectate prin intermediul de legături (*hyperlink*-uri). Aceste resurse sunt denumite pagini web şi conțin *hypertext*, imagini, filme. *Hypertext*-ul este în esență text simplu, la care s-a adăugat o formatare standardizată, ce specifică utilizarea de *hyperlink*-uri. Accesul la resurse se realizează prin intermediul unui

navigator (browser). O pagină web va conține, de obicei, legături (hyperlink-uri) către alte pagini (resurse), justificând denumirea de Web.

World Wide Web-ul este invenția lui **Sir Tim Berners-Lee** din 1989, una din personalitățile marcante în dezvoltarea Internet-ului. Sir Tim Berners-Lee este directorul World Wide Web Consortium (W3C), organizația care coordonează dezvoltarea WWW.

În ultimii ani, WWW-ul a cunoscut o dezvoltare mult mai amplă ajungându-se la folosirea expresiei **Web 2.0** pentru a simboliza serviciile web de nouă generație oferite de diversele comunități web. În același timp, WWW-ul evoluează către ceea ce Sir Tim Berners Lee a denumit **Semantic Web**, adică o extensie a WWW-ului în care conținutul poate fi exprimat în limbaj natural și utilizat de agenți specifici de analiză.

8.4.1 Tehnologiile de bază ale WWW

WWW-ul își datorează existență celor trei tehnologii utilizate de Sir Tim Berners-Lee. Acestea sunt URL, HTTP și HTML și sunt prezentate în continuare.

8.4.1.1 URL

URL înseamnă *Uniform Resource Locator* și este un nume într-o sintaxă bine precizată pentru a identifica resursele din Internet. URL este de fapt un sinonim pentru URI (*Uniform Resource Identifier*).

În mod tipic, un URL/URI este introdus în bara de adrese a unui navigator și este folosit pentru accesarea unei resurse (a unei pagini web) din Internet. Un navigator va „decodifica" URL-ul și va accesa resursa dorită. Astfel, în cazul URL-ul http://www.example.com se face o cerere HTTP către server-ul example.com. Pe de altă parte, URI-ul mailto:bogdan@example.com va însemna deschiderea unui client de e-mail pentru compunerea unui mesaj către bogdan@example.com.

Un URL are o sintaxă bine specificată folosită pentru identificarea stației din Internet, a serviciului folosit de aceasta și a resursei pe care o poate pune la dispoziție acest serviciu.

Astfel, în cazul URL-ului:

http://www.example.com/page.html

stația care deține resursa este server-ul www.example.com. O cerere DNS va fi folosită pentru translatarea numelui într-o adresă IP. Șirul `http://` este folosit pentru identificarea serviciului; în cazul de față acesta este chiar serviciul de Web care folosește protocolul HTTP și portul implicit `80`. Resursa cerută de pe server este `/page.html`. Drept urmare, în urma introducerii URL-ui în bara de adrese a navigatorului, acesta va face o cerere HTTP către stația identificată de numele www.example.com pentru pagina `/page.html` de pe server. Aceasta va fi afișată apoi clientului (în interiorul browser-ului).

Exemplul de URL prezentat mai sus este unul simplificat. Astfel, sintaxa completă pentru un URL este:

```
protocol://server:port/cale/catre/resursa
```

câmpul `protocol` specifică, evident, protocolul folosit pentru comunicație. Acesta poate fi `ftp`, `http`, `scp`, etc.; dacă acest câmp lipsește, un browser va folosi implicit `http`;

câmpul `server` reprezintă numele sau adresa IP a stației cu care se dorește realizarea unei comunicații;

câmpul port este portul utilizat (acesta poate lipsi), caz în care se folosește portul implicit al serviciului;

`/cale/catre/resursa` reprezintă calea către resursa aflată pe server; acesta va folosit această cale pentru a oferi respectiva resursă clientului; de obicei, resursa este o intrare în sistemul de fișiere al server-ului.

Sintaxa URL-ului poate fi complicată. Astfel, un URL tip `ftp` poate asigura inclusiv autentificarea pe server.

Folosirea URL-ului ftp://bogdan:p4r0l48un4@example.com/sources.tgz înseamnă autentificarea utilizatorului `bogdan` cu parola `p4r0l48un4` pe serverul `example.com`, folosind ftp pentru descărcarea fișierului `sources.tgz`.

8.4.1.2 HTTP

HTTP (Hypertext Transfer Protocol) este protocolul fundamental al World Wide Web-ului. Deşi navigatoarele pot folosi alte protocoale în afara HTTP, acesta rămâne cel mai important mecanism de comunicare.

Similar celorlalte servicii din Internet, serviciul Web (HTTP) funcționează ca sistem client-server. Clientul Web este chiar navigatorul (browser-ul) folosit pentru cererea unei resurse prin intermediul unui URL. Portul implicit utilizat de HTTP este 80.

Protocolul HTTP este un protocol de tip întrebare răspuns (*request - response*). Clientul Web (navigatorul) face o cerere către un server, după care acesta îi trimite un răspuns şi resursa cerută. În exemplul de mai jos, telnet este utilizat pentru a efectua o cerere HTTP către `google.ro`:

```
razvan@anaconda:~$ telnet google.ro 80
Trying 72.14.221.104...
Connected to google.ro.
Escape character is '^]'.
GET / HTTP/1.0
HTTP/1.0 302 Found
Location: http://www.google.ro/
Cache-Control: private
[...]
```

După realizarea conexiunii pe portul 80, clientul trimite cererea `GET / HTTP/1.0`. Aceasta înseamnă citirea directorului rădăcină al serverului folosind HTTP versiunea 1.0. Răspunsul server-ului este `HTTP/1.0 302 Found` urmat de pagina efectivă. Dacă pagina dorită nu se află pe server, răspunsul server-ului va fi `Error 402 Page not Found`. Acest răspuns nu înseamnă însă că serverul nu funcționează, caz în care mesajul afişat de client ar fi `Server not Found`.

O altă caracteristică importantă a protocolului HTTP este faptul că este un protocol neorientat pe conexiune (*stateless*). Acest lucru înseamnă că după ce clientul trimite o cerere şi îi soseşte răspunsul conexiunea este încheiată. Pentru o nouă resursă trebuie realizată o altă conexiune. Opus este protocolul FTP care este un protocol cu conexiune persistentă. Petru că este nevoie ca anumite informații să fie persistente de-a lungul mai multor conexiuni s-au găsit diverse soluții: cookie-uri pe client, sesiuni pe server, variabile HTTP, etc.

Un alt neajuns al HTTP este lipsa unui suport puternic de securitate. Acest lucru este corectat prin folosirea protocolului **HTTPS** care foloseşte comunicație criptată. Majoritatea sistemelor de webmail de astăzi folosesc HTTPS pentru a asigura confidențialitatea datelor.

8.4.1.3 HTML

HTML (*Hypertext Markup Language*) este un limbaj folosit pentru a descrie conținutul unei pagini Web. HTML foloseşte **marcaje** (tag-uri) pentru a defini modul în care un element dintr-o pagină Web este afişat de un browser.

O pagină simplă Web descrisă în format HTML este următoarea:

```
razvan@anaconda:~/public_html$ cat main.html
<html>
        <head>
                <title>Pagina mea</title>
        </head>

        <body>
                <h1>Antet</h1>

                Text simplu
                <strong>Text aldin</strong>
                <em>Text cursiv</em>
        </body>
</html>
```

Un marcaj încadrează de obicei un text. La început, marcajul este `<nume_marcaj>` şi se încheie cu `</nume_marcaj>`. Marcajele prezente în această pagină sunt:

- `html`, pentru definirea documentului HTML
- `head` prezintă informaţii despre pagină, spre exemplu titlul (marcajul title)
- `h1` defineşte un antet de nivel 1
- `strong` defineşte un text aldin (bold)
- `em` defineşte un text cursiv (italic)

Un client Web va interpreta o pagină HTML şi o va afişa utilizatorui conform marcajelor existente.

XHTML

XHTML (Extensible Hypertext Markup Language) este un standard şi un limbaj de marcare care are aceleaşi posibilitaţi de exprimare ca şi HTML însă se conformează sintaxei mult mai stricte a XML. Urmând o sintaxă mult mai strictă, un document XHTML este mult mai uşor de interpretat, spre deosebire de un document HTML care necesită un parser complex.

Motivaţia utilizării XHTML o reprezintă creşterea numărului de dispozitive care nu au capacitatea de procesare a unui document HTML complex. Sintaxa strictă a XHTML permite o prelucrare mult mai rapidă a unei pagini Web de către client.

Adoptarea XHTML se realizează într-un ritm inegal datorită lipsei de implementare în navigatoare a caracteristicilor noi pe care standardul le aduce.

Versiunea curentă de XHTML este 1.1. Versiunea 2.0 a standardului este încă în dezvoltare.

8.4.2 Funcţionarea serviciului

După cum s-a precizat, serviciul de Web funcţionează în sistem client-server şi foloseşte protocolul HTTP pentru comunicare. Funcţionarea serviciului este sistematizată în figura de mai jos:

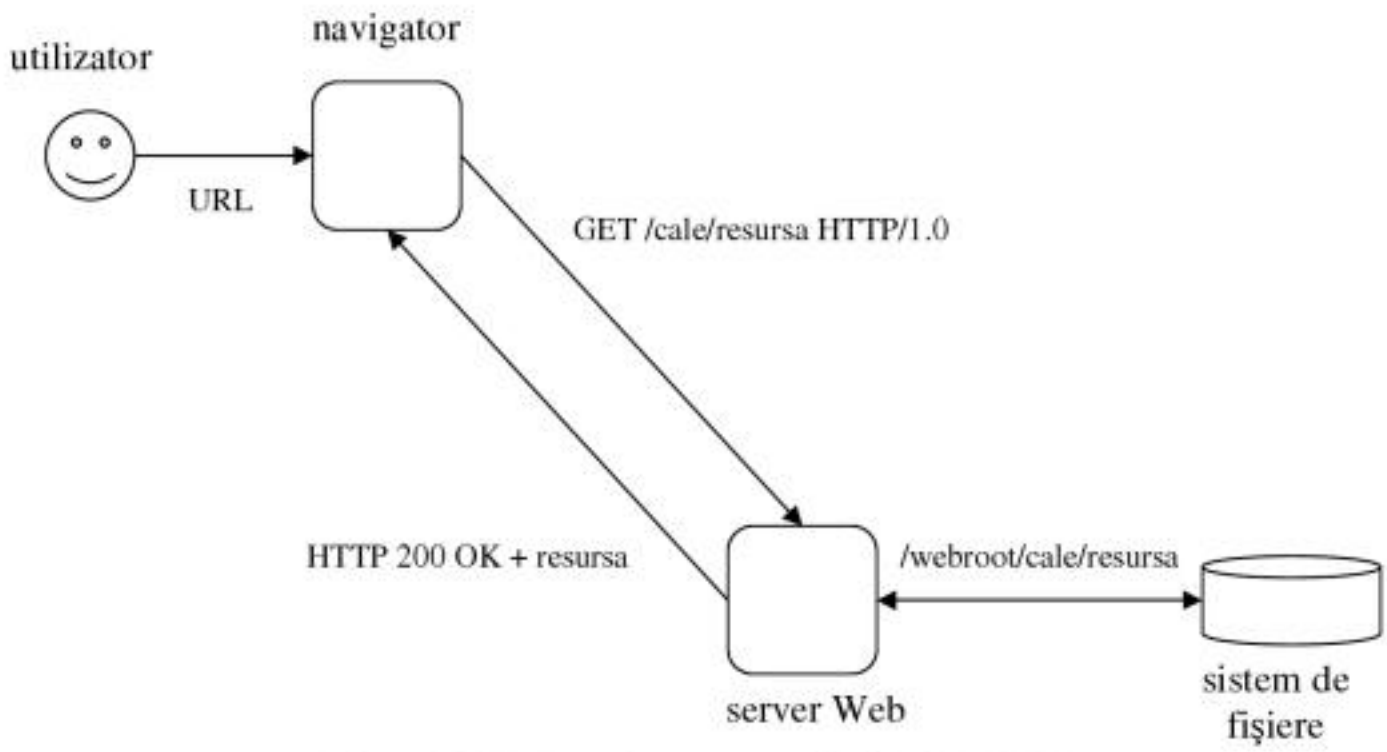


Figura 8-12 Funcționarea serviciului de Web

Figura prezintă un set de paşi urmați tipic în obținerea de către client, a unei resurse aflate pe un server Web (de obicei o pagină Web):

În primă fază utilizatorul va folosi un client Web (navigator) şi va introduce în bara de adrese a acestuia un URL care identifică resursa dorită.

Navigatorul identifică din URL sistemul unde trebuie să se conecteze şi resursa cerută. De obicei se va genera şi o cerere DNS pentru a afla adresa IP asociată serverului. Navigatorul se conectează la server şi, în cazul protocolului HTTP, formulează o cerere de forma `GET /cale/resursa HTTP/1.0`.

Cererea ajunge la serverul Web. Acesta trebuie să ofere clientului resursa cerută. Resursa este, de obicei, o intrare în sistemul de fişiere (pagină web, imagine, film, etc.). Drept urmare serverul Web va trebui să localizeze resursa. Orice server web are configurat un director rădăcină specializat, denumit şi **webroot**. Orice resursă este obținută din acest director. Drept urmare, calea completă în sistemul de fişiere este obținută prin concatenarea căii din cererea HTTP la *webroot*, rezultând intrarea `/webroot/cale/resursa`.

După ce obține resursa obținută, serverul va trebui să o transmită clientului. Pentru aceasta el emite un răspuns `HTTP 200 OK` şi ataşează resursa.

Clientul primeşte resursa de la server şi, dacă este vorba de o pagină Web, interpretează codul (X)HTML şi afişează rezultatul utilizatorului. Conexiunea se închide.

8.4.3 Servere Web

Un **server Web** este responsabil cu oferirea de resurse din sistemul local de fişiere către clienți (navigatoare) prin intermediul protocolului HTTP. Exemple de servere Web sunt Apache HTTP Server, Microsoft IIS, Sun Web Server, Zeus Web Server. Apache HTTP Server este cel mai folosit server Web din Internet.

În general, un server Web prezintă un fişier de configurare prin intermediul căruia îi poate fi alterat comportamentul. În cazul Apache HTTP Server versiunea 2, acest fişier de configurare este `/etc/apache2/apache2.conf`. Alterarea acestui fişier de configurare permite schimbarea portului pe care acesta ascultă conexiuni (implicit 80), a webroot-ului (implicit `/var/www`) şi altele.

Un server Web va primi o cerere HTTP în forma `GET /cale/catre/resursa HTTP/1.0` şi va identifica din sistemul local de fişiere intrarea asociată resursei. Fiecare server are un director rădăcină (denumit *webroot*) unde este căutată resursa pentru a fi oferită clientului. În cazul serverului Apache, în urma unei cereri `GET /cale/catre/resursa HTTP/1.0` va trimite clientului fişierul identificat de `/var/www/cale/catre/resursa` (sau `/usr/local/Apache2/htdocs`).

8.4.4 Clienţi Web

Clienţii Web se numesc **navigatoare** sau **browser-e**. Un client Web se va conecta la un server Web pentru a solicita o resursă. Resursa este specificată de utilizator într-un URL. Clientul Web transmite o cerere HTTP către server solicitând resursa clientului. De obicei această resursă este o pagină Web. Codul HTML este interpretat apoi de navigator şi afişat utilizatorului.

Cele mai utilizate tipuri de navigatoare sunt Internet Explorer, Mozilla Firefox, Opera şi Safari. Internet Explorer este cel mai utilizat navigator cu o pondere de 80-85% urmat de Mozilla Firefox cu o pondere de 10-15%. Celelalte navigatoare au ponderi mult mai mici. Există şi navigatoare în linie de comandă, cum ar fi lynx, w3m, links.

Navigatoarele Web reprezintă una dintre cele mai utilizate aplicaţii din cadrul unui sistem de operare. Competiţia dintre acestea a generat ceea ce s-a numit „*browser wars*". Astfel, anii 1994-1998 au fost marcaţi de competiţia dintre Netscape Navigator şi Internet Explorer cu victoria celui din urmă. În anul 2002, Internet Explorer era folosit în proporţie de circa 92%. Ultimii ani sunt dominaţi de competiţia între Mozilla Firefox şi Internet Explorer.

Competiţia dintre navigatoare a condus şi la apariţia unui număr important de caracteristici printre care: navigare tabulară, blocarea mesajelor de tip pop-up, corecţie gramaticală, gesturi de mouse, suport pentru diverse protocoale (FTP, IRC, BitTorrent) şi pentru standarde (CSS, XHTML, XSLT, RSS).

8.5 Partajarea de fişiere

8.5.1 SMB

SMB (*Server Message Block*) este un protocol de nivel aplicaţie, care face posibilă partajarea de resurse peste conexiunea TCP/IP. Creat iniţial de IBM, acest protocol a fost adoptat de Microsoft şi a suferit până în prezent numeroase transformări, atât ca funcţionalitate cât şi ca specificaţii de bază. În prezent, SMB a ajuns la versiunea 2.0 în cadrul distribuţiei Windows Vista, prezentând îmbunătăţiri majore, acestea incluzând o utilizare mai eficientă a lăţimii de bandă şi o complexitate mai scăzută a implemenetării.

Precum majoritatea protocoalelor folosite în reţele, SMB este un protocol care funcţionează conform modelului client-sever. Astfel, implementarea constă în instalarea de servicii/resurse (server de fişiere, imprimantă) pe un calculator pe care rulează SMB server şi partajarea acestora pe restul calculatoarelor din reţea, pe care rulează clienţi SMB.

Datorită modificărilor pe care Microsoft le-a adus protocolului original, a existat bineînţeles problema compatibilităţii cu celelalte variante ale SMB din alte sisteme de operare. Astfel s-a născut soluţia gratuită numită **Samba**. Distribuită sub licenţă publică GNU, Samba este o reimplementare a SMB/CIFS care oferă interoperabilitate cu varianta protocolului folosită pe Windows, şi care rulează în majoritatea mediilor Unix/Linux, BSD sau chiar MAC OS X Server. Samba a rezolvat problema partajării resurselor în aceeaşi reţea între o staţie Windows şi o staţie Linux. Putem astfel folosi de pe staţia Linux, funcţia de imprimare de pe staţia Windows, sau accesa de pe staţia Windows, fişiere de pe staţia ce rulează Linux.

Există patru aplicații de bază pentru Samba:

- partajarea unei partiții Linux cu o stație Windows
- accesarea din Linux a fișierelor partajate prin SMB
- partajarea imprimantei din Linux
- partajarea imprimantei din Windows

Instalarea și configurarea Samba este prezentată în studiul de caz al acestui capitol.

8.5.2 FTP

FTP (file transfer protocol) este un protocol foarte răspândit ce oferă posibilitatea transferului de fișiere atât într-o rețea locală cât și prin Internet. Se bazează pe modelul client-server, deci presupune existența a două entități între care se realizează un transfer de fișiere, clientul fiind cel care se conectează (și se autentifică) la serverul de la care poate descărca sau căruia îi poate trimite fișiere.

Ca și în cazul modelului generalizat de client-server, serviciul FTP oferit de către acesta din urmă se reduce la existența unei aplicații, numită "server FTP" ce rulează pe o stație. Similar, clientul rulează o aplicație numită "client FTP" ce se ocupă de managementul conexiunii către server și de traficul de date dintre aceștia. Atât aplicațiile server cât și cele client FTP sunt disponibile, practic, pentru toate platformele existente, dovadă a versatilității și răspândirii protocolului FTP.

Un dezavantaj al protocolului FTP este faptul că acesta transferă atât informațiile de conectare cât și conținutul propriu-zis al fișierelor în mod necriptat, interceptarea unui astfel de trafic putând duce la o interpretare facilă a sa.

8.5.2.1 Funcționare

Un client FTP ce se conectează la un server trimite acestuia un set de date de autentificare (nume și parolă) care au asociate pe server un o structură de directoare ce sunt puse la dispoziția acelui utilizator. Dacă autentificarea reușește, clientul are dreptul să trimită în continuare comenzi pentru a interacționa cu fișierele la care are acces. Drepturile de acces la fișiere se reduc la cele de listare/creare de directoare, citire, modificare, ștergere de fișiere sau directoare, ceea ce implică un transfer bidirecțional între client și server (clientul poate atât să descarce, cât și să trimită fișiere pe server). În practică, aceste drepturi sunt enunțate în aceeași structură de tip "`rwx`" ca și pe sistemele UNIX.

Serverul FTP "ascultă" conexiunile pe portul 21 și realizează transferul de comenzi tot pe acesta. Transferul efectiv de fișiere se realizează în paralel, pe portul 20, doar în urma unei conexiuni prestabilite. Porturile rezervate pentru protocolul FTP sunt 21 pentru stabilirea conexiunii și 20 pentru transferul efectiv de date. O variantă securizată și similară o reprezintă SFTP (SSH File Transfer Protocol). O altă limitare a sa este imposibilitatea de a transmite atributele ce țin de data și ora creării/modificării fișierelor, acestea fiind rescrise la destinație. De asemenea, această limitare este eliminată în cazul lui SFTP.

Transferul datelor prin FTP se poate face prin două moduri, selectate manual înaintea inițierii transferului sau determinate automat de către clientul FTP:

Mod ASCII: folosit pentru transferul fișierelor text; are avantajul de a transmite datele folosind codurile ASCII ale caracterelor, ceea ce permite compresia dinamică a informației, la un raport foarte bun, deci un flux mai puternic de date, dar are dezavantajul de a corupe conținutul oricărui fișier non-text.

Mod binar: folosit pentru transferul oricărui tip de fișiere (inclusiv text)

8.5.2.2 Exemplu de utilizare

Conectarea la un server FTP dintr-o linie de comandă Linux se poate face direct prin comanda "ftp", urmată de numele sau adresa serverului la care se doreşte conectarea, ca în exemplul:

```
[rookie@localhost ~]$ ftp ftp.lug.ro
Trying 193.226.140.51...
Connected to ftp.lug.ro (193.226.140.51).
220 (vsFTPd 2.0.3)
Name (ftp.lug.ro:root): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Promptul "`ftp>`" indică faptul că următoarele comenzi vor fi trimise direct către server. Câteva comenzi comune sunt `ls` sau `dir` pentru a lista conținutul directorului curent, `cd` pentru a schimba directorul, `get` pentru a descărca fişierele specificate ca parametri (`mget` pentru fişiere multiple) sau `put` pentru trimiterea de fişiere. Deconectarea de la serverul FTP se face prin comenzile `quit` sau `disconnect`.

O categorie specială de clienți FTP o reprezintă broswerele web, limitarea acestora constând în faptul că nu permit decât descărcarea fişierelor de pe server. Conectarea la un server FTP printr-un browser web se poate face prin introducerea adresei sale în bara de adrese a browserului, specificându-se protocolul:

```
ftp://ftp.lug.ro
```

În cazul serverelor în care accesul nu se poate face anonim (fără cont predefinit), browserul va cere un nume de utilizator şi o parolă.

Pentru a specifica direct din adresa parametrii de autentificare, se poate scrie:

```
ftp://user:parola@server:2555
```

Exemplul anterior specifică explicit şi portul pe care se realizează conexiunea (2555), informație necesară în cazul în care serverul nu rulează pe portul implicit, 21.

8.5.3 Rețele de partajare de fişiere

Partajarea fişierelor (*file sharing*) a devenit una dintre cele mai răspândite utilizări ale Internetului din prezent. Fenomenul se regăseşte în rețelele în care utilizatorii au dreptul de a pune la dispoziție propriile fişiere, de a căuta şi de a descărca fişierele altora. În general, conceptul de partajare de fişiere este considerat neintruziv şi totodată aproape lipsit de costuri pentru cel care îşi expune fişierele din moment ce copiile sale rămân nealterate, iar singurul drept al utilizatorilor cu care le partajează este de a le vizualiza şi de a le descărca.

Fişierele ce sunt puse la dispoziția altor utilizatori prin intermediul rețelelor de partajare de fişiere pot reprezenta, practic, orice fel de informații, de la creații proprii şi fişiere personale, până la diverse variante de software distribuit doar sub licență, muzică, filme, etc. Diversitatea datelor ce sunt puse la dispoziție prin *file sharing* şi lipsa practică a oricăror limitări sau reglementări de natură legală au transformat concepția generală despre rețelele de *file sharing* într-o formă de facilitare a distribuirii de informații ilegale. Mai mult chiar, au dus la asocierea implicită a rețelelor de partajare de fişiere cu transferul de copii ilegale ale datelor.

Într-adevăr, o mare parte dintre rețelele de partajare de fişiere au luat naştere tocmai cu acest scop în spate, de distribuire ilegală a informației comerciale, dar o bună parte dintre ele au rezistat presiunilor legale tocmai pe baza faptului că rețeaua în sine şi tehnologiile folosite nu

au nici o legătură cu scopurile ilegale în care sunt utilizate. Din punct de vedere legal, o reţea de *file sharing* există doar prin prisma utilizatorilor ei şi a fişierelor pe care aceştia sunt dispuşi să le ofere; de aceea, în ultimă instanţă, responsabilitatea revine utilizatorului unei astfel de reţele. Utilizarea în sine a unei reţele de *file sharing* nu este ilegală, ci doar copierea şi distribuirea informaţiilor aflate sub protecţia legii copyright-ului. În ultimă instanţă, acţiunile legale ce au fost luate asupra reţelelor de partajare de fişiere au avut la bază argumente conform cărora astfel de reţele facilitau răspândirea copiilor ilegale.

8.5.3.1 Scurt istoric

Reţelele de partajare de fişiere au luat naştere sub forma unui model client-server, ceea ce implica o structură centralizată iniţial a fişierelor propriu-zise iar apoi doar a listei fişierelor disponibile pe care utilizatorii o accesează în momentul în care caută anumite fişiere, transferul efectiv făcându-se după stabilirea unei conexiuni peer-to-peer (P2P) între cel ce deţine fişierul şi cel ce l-a cerut. Deşi tendinţa reţelelor de *file sharing* duce spre descentralizare, câteva dintre reţelele modelului client-server funcţionează şi astăzi. Două dintre primele şi cele mai remarcabile exemple au fost Napster (www.napster.com) şi Audio Galaxy (www.audiogalaxy.com) ambele axate pe distribuţia de muzică. Caracterul gratuit al ambelor a fost desfiinţat, primul în 2001 şi cel de-al doilea în 2002, în prezent ele continuând să ofere melodii spre download publicului, dar contra cost. De asemenea, reţeaua Kazaa (www.kazaa.com) folosită în principal pentru distribuţia de filme şi muzică a reprezentat un fenomen ce a luat amploare foarte rapid şi s-a menţinut până în 2005, când i s-a interzis traficul ilegal iar popularitatea i-a scăzut drastic atât datorită deciziilor legale luate, cât şi datorită faptului că utilizatorii erau nevoiţi să folosească un client ce număra nu mai puţin de opt programe malware (*adware* şi *spyware*). Kazaa a reprezentat totuşi primul pas spre descentralizare, utilizatorilor cerându-li-se doar să se autentifice pe un server central.

În prezent, cea mai mare parte din traficul de *file sharing* (şi totodată un segment considerabil al traficului P2P) din Internet se împarte între protocoalele DC (Direct Connect) şi BitTorrent, ambele utilizând programe-client dedicate.

8.5.3.2 Direct Connect

Caracterul descentralizat al protocolului Direct Connect permite utilizatorilor să se conecteze la un hub (practic un server, pe care utilizatorul îl vede ca pe un chatroom) a cărui principală funcţie este de a redirecţiona cererile de căutare de fişiere pe care orice utilizator le poate efectua, către toţi ceilalţi utilizatori conectaţi. Adresele huburilor pot fi introduse manual sau alese din "hublist"-uri conţinute în majoritatea clienţilor. In prezent, numărul huburilor publice disponibile este de ordinul zecilor de mii, fiecare cu până la câteva mii de utilizatori conectaţi. Unul dintre avantajele protocolului DC constă în faptul că oricine îşi poate instala software-ul necesar pentru rularea unui astfel de hub în reţeaua privată sau chiar peste Internet.

Clienţii DC sunt open-source, ceea ce a permis apariţia unei mari varietăţi de clienţi, fiecare cu facilităţile sale. La baza acestora stă clientul original, denumit DC++ (dcplusplus.sourceforge.net), scris în C++. Printre alţi clienţi derivaţi şi răspândiţi se numără: oDC, Apex DC++, LDC++, zDC++, rmDC++, StrongDC iar facilităţile lor includ: management mai eficient şi mai complex al sarcinilor de descărcare sau trimitere (a traficului, în general) a fişierelor partajate, control îmbunătăţit asupra interacţiunii cu ceilalţi utilizatori, posibilitatea de segmentare a fişierelor (descărcare simultană de la mai multe surse), algoritmi de verificare a integrităţii fişierelor (*hash check*) şi de filtrare a fişierelor false (*fake share*), prioritizarea traficului, protecţie împotriva coruperii fişierelor, etc. De asemenea, clienţii DC permit căutarea fişierelor pe baza codurilor hash pentru a identifica în mod corect şi unic fişierele căutate.

Utilizatorii huburilor DC pot pune la dispoziţie practic orice fişiere, fără a se ţine cont de tipul, mărimea, originea (aflate sub copyright sau nu) lor. Majoritatea huburilor cer utilizatorilor să ofere o cantitate minimă de astfel de fişiere doar pentru a obţine dreptul de se conecta la hub.

Această limită impune automat şi o cantitate mare de date disponibile prin intermediul utilizatorilor unui hub.

Deşi Linux oferă o varietate mare de clienți DC, atât în linia de comandă cât şi cu interfață grafică, capabilitățile acestora sunt limitate în comparație cu echivalentele lor construite pentru Windows (fapt valabil si pentru restul rețelelor de partajare de fişiere), mai ales din cauza unei mari concentrări a utilizatorilor acestor rețele în mediul Windows.

Exemplu de configurare: Valknut (http://packages.debian.org/stable/net/valknut) – varianta mai recentă a vechiului pachet `dcgui`.

Figura 8-13 Valknut. Lista de huburi returnată din hublist-ul implicit, ordonate după numărul de utilizatori

"*Hublist*"-ul poate fi descărcat din ecranul listelor de huburi (butonul "*Update public hub list*"). Tot de aici se poate intra, printr-un dublu clic, pe hubul dorit.

În fereastra de opțiuni se recomandă setarea unui nume de utilizator (nickname) în forma [Țară][Oraş][ISP]Username. Sub tabul "*Hub Lists*" se pot introduce mai multe adrese ale unor liste de huburi.

Sub tabul "*Transfers*" utilizatorul trebuie să specifice o directoarele ce le va partaja în toate huburile la care se va conecta. Aceasta este şi o condiție pentru conectarea la majoritatea huburilor. În medie, cantitatea minimă de date partajate oscilează între 1 GB şi 30 GB pentru huburile publice şi poate fi aflată din descrierea hubului din hublist (înainte de conectare) sau dintr-un mesaj de eroare primit de la hub în urma unei conectări eşuate.

Sub tabul "*Connection*" se alege modul de conectare: "*Active*" sau "*Passive*". Într-o rețea locală cu adrese private (ref. 7.2.1) în general trebuie selectată opțiunea "*Passive*". Rezultatele căutărilor în huburi sunt returnate printr-un port UDP, iar transferul efectiv de date se face printr-un port TCP, ceea ce implică accesul din exterior la aceste porturi fie printr-o conexiune directă la Internet, fie prin *port forwarding* (en.wikipedia.org/wiki/Port_forwarding), acesta fiind modul de funcționare "Active". În cazul în care aceste porturi nu pot fi accesate direct din Internet, clientul funcționează în mod "*Passive*", adică nu poate primi date decât prin conexiunile inițiate de el însuşi. În această situație, vizibilitatea clientului va fi limitată doar la utilizatorii activi, pentru că nu va putea contacta explicit pe cei pasivi şi nici aceştia nu-l vor putea contacta înapoi.

Odată conectat la un hub, utilizatorul poate să descarce listele fişierelor partajate (*filelists*) de ceilalți utilizatori din hub din care poate selecta fişierele pe care doreşte să le descarce. Utilizatorul nu trebuie să aprobe sau să supravegheze fişierele în care sunt descărcate de la el însuşi de către ceilalți utilizatori, trimiterea ulterioară a fişierelor şi managementul conexiunilor realizându-se automat.

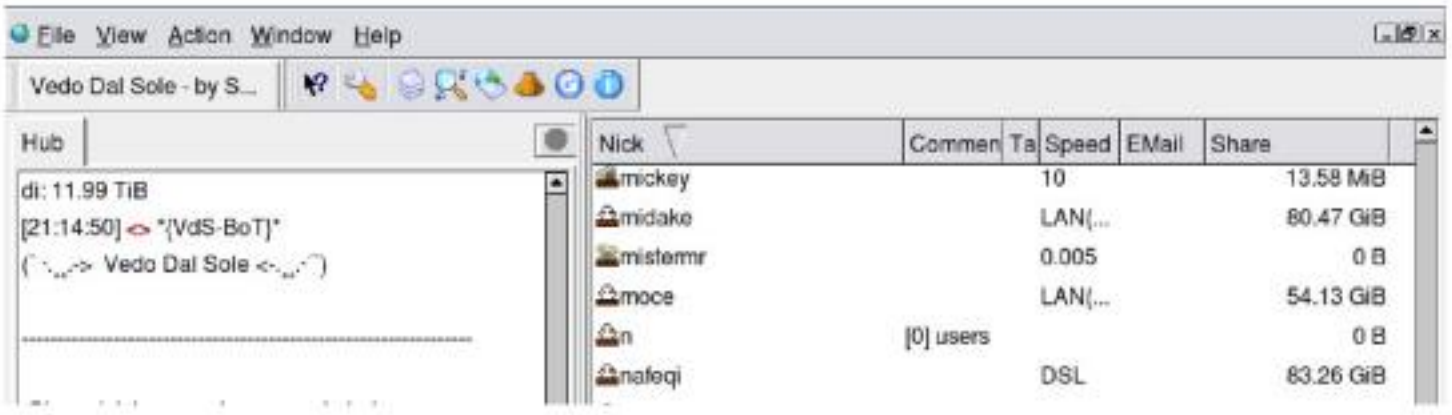


Figura 8-14 Valknut. Utilizatori accesibili în urma conectării la un hub

8.5.3.3 Bittorrent

Protocolul Bittorrent a cunoscut una dintre cele mai spectaculoase răspândiri din istoria rețelelor de partajare de fișiere. Modul său de acces la fișiere presupune accesul la un site web central (denumit "*tracker*"), care ține evidența tuturor utilizatorilor conectați în acel moment și a fișierelor ce le partajează. Pentru a descărca aceste fișiere, se descarcă mai întâi un fișier-rezumat care conține informații despre fișierele selectate, precum și o adresă folosită de client prin care acesta obține lista utilizatorilor ce le partajează. Acesta are extensia ".torrent" și poate conține informații despre o multitudine de fișiere, eventual structurate pe directoare, precum și informații de verificare a integrității fișierelor ("*hash check*"; o formă frecventă sunt fișierele .sfv = "*simple file verification*"). După obținerea coordonatelor utilizatorilor de la care poate descărca fișierele respective, clientul inițiază conexiuni punct la punct (*peer-to-peer*) cu fiecare dintre aceștia, descărcând simultan diferite segmente ale fișierelor.

Unul dintre marile avantaje ale Bittorrent-ului îl reprezintă vitezele foarte mari ce pot fi atinse datorită posibilității de a descărca de la foarte mulți utilizatori simultan. De asemenea, partajarea fișierelor nu se face doar pe fișiere integrale, ci pe mici segmente, ceea ce înseamnă că din momentul în care un utilizator a obținut un segment dintr-un fișier (ce poate fi între 32 KB și 4 MB) acel segment este automat partajat celorlalți utilizatori, asigurându-se astfel o răspândire exponențială, foarte rapidă a datelor, chiar și în cazul în care acestea au originat dintr-o singură sursă.

Partajarea unor fișiere noi se face anunțând tracker-ul, după ce utilizatorul a generat fișierul .torrent corespunzător pentru a putea oferi celorlalți utilizatori acces. Acest mod de notificare și management al utilizatorilor a creat din Bittorrent o rețea bazată pe ordine și integritate, fiind obligatorii respectarea anumitor standarde stricte înaintea partajării fișierelor. De aceea, nu e deloc surprinzător faptul că majoritatea distribuțiilor de Linux, spre exemplu, pot fi descărcate legal și eficient prin rețeaua Bittorrent, acest mod fiind deseori recomandat pentru a ușura traficul pe serverele de distribuții.

Clienții de Bittorrent sunt extrem de numeroși, cu diverse capabilități. Primul client a fost cel ce a păstrat numele protocolului, Bittorrent, și este în continuare disponibil atât pentru Windows, cât și pentru Linux.

Exemplu de configurare: Azureus (http://azureus.sourceforge.net/)

Configurarea inițială a lui Azureus se poate face în funcție de nivelul de expertiză al utilizatorului.

- *Beginner* = nivel începător, acces doar la descărcarea de torente
- *Intermediate* = nivel mediu, acces la funcțiile tracker-ului, oferă posibilitatea de a crea torente
- *Advanced* = acces la setări avansate de rețea, control asupra conexiunilor

Odată ce fișierul .torrent a fost obținut, acesta poate fi încărcat în Azureus iar descărcarea conținutului său va începe automat. Pot fi descărcate simultan fișiere obținute din mai multe fișiere .torrent, lista acestora fiind vizibilă în partea superioară a clientului. În momentul în care conținutul unui fișier .torrent s-a terminat de descărcat, acesta va fi automat trecut în starea de

"*seeding*", în care aşteaptă conexiuni din exterior pentru a servi acele fişiere altor utilizatori. Atenţie, ştergerea unei sarcini în curs de descărcare trebuie făcută având în vedere, explicit, fişierele parţiale descărcate până în acel moment (clic dreapta > "*Remove*" > "*Delete Data/Both*"). Similar, eliminarea unei sarcini care a fost complet descărcată nu ar trebui făcută nici împreună cu şi nici doar cu ştergerea efectivă a datelor de pe disc.

Figura 8-15 Azureus. Exemplu de torent în curs de descărcare

Pentru a vizualiza opţiuni detaliate pentru fiecare sarcină de descărcare, se poate da dublu clic pe numele unui *torrent* în curs de descărcare. Informaţiile suplimentare conţin lista integrală a utilizatorilor cu care s-au realizat conexiuni şi stările lor, starea *tracker*-ului, informaţii legate de volumul de date transferat, rata de eroare, detaliile segmentării fişierelor, etc.

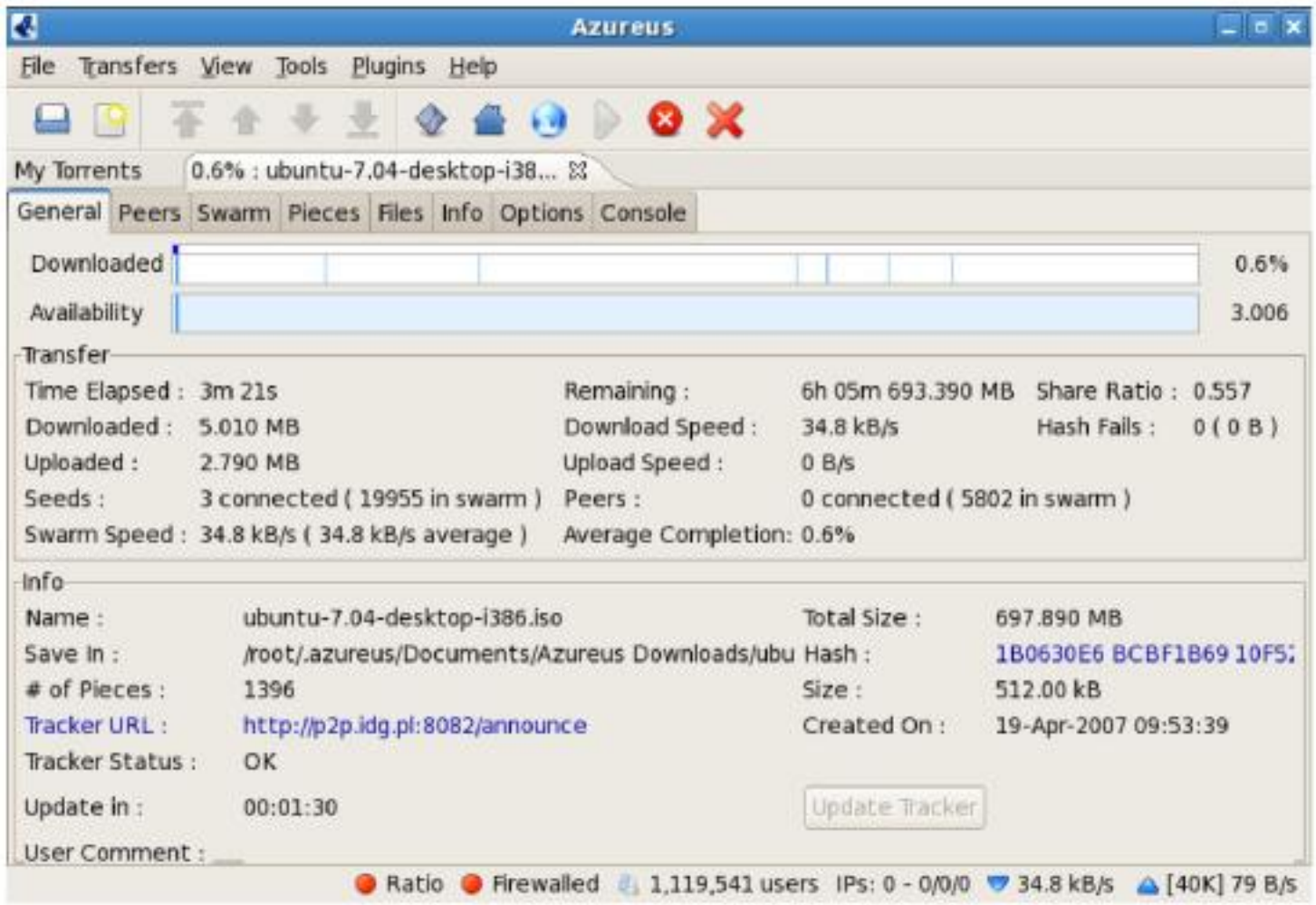


Figura 8-16 Azureus. Detalii despre un torent în curs de descărcare

Crearea unui nou fişier .torrent se face din meniul "*File*" > "*New Torrent*". Principala opţiune este, bineînţeles, fişierul sau directorul ales drept conţinut de partajat pentru torrentul respectiv. Un exemplu de utilizare a aplicaţiei este disponibil la adresa http://paulstamatiou.com/2006/01/04/how-to-make-a-torrent-with-azureus

O mare parte dintre *tracker*-ele disponibile cer autentificarea utilizatorilor ce accesează fişierele prin intermediul lor. Mai mult, ele ţin evidenţa traficului primit şi trimis de către utilizatori, având libertatea de a impune anumite criterii cantitative cu privire la trafic, ceea ce se traduce de cele mai multe ori în "*ratio*", adică raportul dintre volumul de date trimise şi primite. Prin aceste reglementări se asigură o disponibilitate sporită a datelor, asigurând viabilitatea surselor un timp cât mai îndelungat după publicarea torentului şi răspândirea sa. Acest raport poate fi vizualizat şi în Azureus, sub coloana "*Share ratio*". Un raport supraunitar este suficient pentru condiţiile oricărui *tracker*.

8.6 Studiu de caz:

8.6.1 Configurarea Samba

Modul de instalare Samba şi de obţinere a partajării de fişiere între cele doua sisteme de operare este descris în cele ce urmează.

Instalarea se poate face folosind sursa programului, sau utilitarul `apt`:

```
kiwi:~$ sudo apt-get install samba
```

Aplicaţia foloseşte un fişier de configurare numit `smb.conf`, pe care îl citeşte şi îl încarcă de fiecare dată când porneşte.

Odată instalată, Samba îşi va crea acest fişier de configurare, predefinit, în `/etc/samba/`. În această formă predefinită, acest fişier nu asigură funcţionalitate aplicaţiei, ci mai degrabă oferă directive de protocol şi instrucţiuni de configurare.

Pentru că acest fişier, în formă iniţială, nu este de prea mult ajutor, îl vom păstra redenumit pe disc (pentru a-l folosi ulterior pentru indicaţii şi directive) şi vom crea de la zero un nou fişier de configurare, pe care Samba îl va folosi pentru partajarea dintre cele două sisteme. Mai întâi este redenumit fişierul iniţial, apoi creat un nou fişier de configurare:

```
kiwi:/etc/samba$ sudo mv smb.conf smb.conf.initial
kiwi:/etc/samba$ sudo touch smb.conf
kiwi:/etc/samba$ sudo vim smb.conf
```

Pentru început, se vor adăuga în fişierul de configurare directive globale, care reprezintă configurări valabile pentru toate resursele partajate. În continuare sunt specificate liniile ce trebuie introduse în fişier, alături de comentarii (marcate cu ';') asociate fiecărui parametru.

```
[global]
; parametrii globali de configurare
netbios name = waters-laptop
; aici trebuie introdus numele staţiei Linux
; numele staţiei se poate afla prin comanda de consolă: hostname
workgroup = RETEA
; acesta este 'workgroupul' definit în Windows
announce version = 5.0
socket options = TCP_NODELAY
passdb backend = tdbsam
; setarea opţiunilor de socket şi specificarea tipului de baza de date
; folosit pentru păstrarea parolelor, în cazul folosirii autentificării
security = user
; dacă această directivă este prezentă, fiecare utilizator care accesează
; resursele partajate va avea nevoie de un cont pe staţia Unix prin care
; să facă această operaţie
null passwords = true
; specifică posibilitatea de a folosi utilizatori samba, fără
; autentificare cu parolă
name resolve order = hosts wins bcast
```

```
; specifică ordinea traducerii numelor folosite pe rețea, în cazul în care
; resursa partajată
; se accesează cu un nume în loc de adresă IP
wins support = yes
; activarea WINS pentru a putea referi stațiile din rețea prin nume,
; în loc de adresa IP
syslog = 1
syslog only = 1
```

După ce s-au stabilit configurațiile globale, se adaugă în `smb.conf` şi directivele pentru resursa partajată.

Pentru a configura schimbul de fişiere cu o stație Windows se adaugă următoarele linii de cod:

```
[MyShare]
; numele dat resursei pe care o partajăm
; "nume stație" este numele configurat mai sus la directiva netbios name
path = /home/waters/share
; setarea căii spre directoarele sau fişierele pe care vrem sa le partajăm
; putem scrie calea inclusiv spre o partiție montată în Linux
; (Ex: /mnt/hda2 sau /media/sda1)
read only = no
; această opțiune oferă utilizatorilor dreptul de scriere în propriul
; director home
guest ok = no
; acest parametru determină dacă se poate efectua autentificarea ca guest
create mask = 0600
; create mask reprezintă umask-ul pentru fişierele care sunt create pe
; calculatorul accesat la distantă, de către utilizatorul care accesează
; fişierele partajate
directory mask = 0755
; directory mask este umask-ul pentru directoarele din resursa partajată
```

8.6.1.1 Crearea de utilizatori Samba

Toate configurațiile de care are nevoie serverul Samba se fac în `smb.conf`. După ce acest fişier a fost editat precum în exemplul de mai sus, tot ceea ce rămâne de făcut este adăugarea utilizatorilor sistemului ca utilizatori Samba.

Adăugarea utilizatorilor reali ca utilizatori Samba, se face comunicând aplicației, care dintre utilizatorii sistemului Linux vor participa la schimbul de fişiere şi vor fi folosiți în autentificarea de pe o maşină Windows. Pentru această operație, se foloseşte utilitarul `smbpasswd`, care se instalează odată cu Samba. Dacă vrem să configurăm utilizatorul `waters`, ca fiind un utilizator Samba procedăm astfel:

```
kiwi:/$ sudo smbpasswd -L -a waters
kiwi:/$ sudo smbpasswd -L -e waters
```

Parametrul `-a` specifică crearea utilizatorului, iar parametrul `-e` specifică activarea utilizatorului. La crearea utilizatorului Samba, numit `waters`, sistemul va cere configurarea unei parole pentru utilizator.

Dacă se doreşte o securitate sporită, se va crea mai întâi un utilizator nou, căruia nu i se dă dreptul de a accesa linia de comandă, iar apoi se va configura acest utilizator ca fiind un utilizator Samba.

```
kiwi:/$ sudo useradd -s /bin/false user23
kiwi:/$ sudo smbpasswd -L -a user23
kiwi:/$ sudo smbpasswd -L -e user23
```

După ce cel puțin un utilizator Samba a fost creat şi activat, daemonul Samba se poate porni astfel:

```
root@waters-laptop:/# etc/init.d/samba start
```

În continuare este prezentat modul de accesare a fişierelor partajate, atât din Windows cât şi din Linux şi activarea opţională a WINS pe un sistem de operare Windows.

În primul rând, pe Windows, va trebui sa fie setat acelaşi workgroup ca şi în fişierul de configurare smb.conf (în cazul de faţă: **RETEA**). Pe Windows workgroupul iniţial este **MSHOME**. Pentru a-l schimba, configuraţiile sunt după cum urmează:

```
  > "Select" > "Start" > Clic dreapta "My Computer" > Clic pe tabul
"Computer name" > Clic pe "Change" > Schimbare workgroupul în RETEA > Clic
"Apply"
```

Pe staţia Windows, accesarea fişierelor partajate se poate face astfel:

```
  > "Select" > "Start" > Clic dreapta "My Computer" > Selectarea "Map
network drive" > Alegerea unei litere care să reprezinte map-ul > Scrierea în
dreptul Folder: \\adresă IP\nume_share > "Finish"
```

Utilizarea serviciului WINS, oferă facilitatea accesării unei resurse partajate prin numele staţiei, în loc de adresa sa IP. Dacă se doreşte folosirea WINS (s-a setat directiva **wins support = yes** în fişierul de configurare Samba), următoarele configuraţii trebuie făcute în Windows:

```
  > "Select" > "Start" > "Control Panel" > "Network Connections" > "LAN
Connection" (legătură de reţea prin care se face comunicarea cu staţia Linux)
> "Properties" > selectare "(Internet Protocol) TCP/IP" din lista şi clic
"Properties" > "Advanced" > Selectare tab "WINS" > "Add" > Scriere adresă IP
a staţiei Linux > "Add" > Selectare "Enable NetBIOS over TCP/IP" > "OK" >
"OK" > "Close" > Repornire Windows
```

Figura 8-17 Activarea serviciului de WINS

Dacă WINS a fost activat, accesarea resurselor partajate se face ca mai sus, cu menţiunea că în locul adresei IP se poate scrie numele staţiei Linux. După ce se apasă butonul "*Finish*", utilizatorului Windows i se va cere un nume de utilizator şi o parolă pentru accesarea fişierelor. Trebuie introduse numele de utilizator Samba configurat cu `smbpasswd` şi parola pentru acesta.

Accesarea fişierelor partajate de pe staţia Linux se poate face din mediul grafic sau din consolă.

Din mediul grafic, fişierele se pot accesa folosind Konqueror (pentru KDE) sau Nautilius (pentru Gnome), prin introducerea adresei smb în bara de adrese:

```
smb://nume staţie/fişier.
```

Din consolă, fişierele de pe Windows se pot accesa prin folosirea `smbclient`, instalat odată cu Samba. Spre exemplu, dacă avem serverul de WINS activ comanda este:

```
kiwi:/$ smbclient //andreixp/temp -U Andrei
```

Fără WINS activat:

```
kiwi:/$ smbclient //192.168.2.101/temp -U Andrei
```

- `andreixp` reprezintă numele staţiei Windows; dacă WINS nu funcţionează, în acest câmp trebuie trecută adresa IP a staţiei Windows, precum în al doilea exemplu.
- `temp` reprezintă directorul partajat; dacă din comandă lipseşte acest parametru, se vor accesa toate directoarele puse în comun pe staţia Windows.
- `Andrei` reprezintă numele utilizatorului Windows pe contul căruia sunt accesate fişierele.

După executarea comenzii, utilizatorului Linux i se va cere parola utilizatorului Andrei. Daca acesta nu are configurată o parolă, se poate apasă direct tasta <ENTER>. În ambele cazuri, linia de comandă se va transforma într-un prompt în care se vor putea executa comenzi de listare de fişiere, descărcare de fişiere sau ieşire din clientul SMB.

```
root@kiwi:/# smbclient //andreixp/temp -U Andrei
Password:
Domain=[ANDREIXP] OS=[Windows 5.1] Server=[Windows 2000 LAN Manager]
Listarea conţinutului directorului temp se face prin comanda ls
smb: \> ls
  .                                   D        0  Sat Jun  9 21:27:30 2007
  ..                                  D        0  Sat Jun  9 21:27:30 2007
  000_0007.jpg                        A   331764  Sat Jun  9 21:20:51 2007
  000_0008.jpg                        A   322224  Sat Jun  9 21:20:52 2007

                37056 blocks of size 1048576. 3984 blocks available
```

Pentru a descărca un fişier, se utilizează comanda `get`:

```
smb: \> get 000_0010.jpg
getting file \000_0010.jpg of size 340008 as 000_0010.jpg (4049.3 kb/s)
(average 4049.3 kb/s)
```

Pentru afişarea tuturor comenzilor disponibile, se poate folosi comanda `help`. Ieşirea din clientul SMB se face prin comanda `q`:

```
smb: \> q
kiwi:/$
```

Cuvinte cheie

- Bittorrent
- client
- curl
- Direct Connect
- FTP
- HTML
- HTTP
- IMAP
- IP
- LDA
- POP3
- port
- punct-la-punct
- scp
- server
- SMB
- spam
- SSH (Secure Shell)
- stivă de protocoale
- TCP
- telnet
- torrent
- UDP
- URL
- webmail
- wget
- WWW
- XHTML

Întrebări

Care dintre operațiile de mai jos nu se bazează pe o arhitectură client-server?

- ridicarea mesajelor
- editarea unui nou mesaj
- trimiterea unui nou mesaj
- verificarea mesajelor folosind broswer

Protocolul telnet comunică direct (fără intermedierea nivelului transport) cu nivelul rețea. După stabilirea unei conexiuni de telnet comunicația se va face doar unidirecțional: de la server către client.

- adevărat, adevărat
- adevărat, fals
- fals, adevărat
- fals, fals

Care dintre serviciile de mai jos nu este util intr-o soluție de webmail?

- SMTP
- IMAP
- HTTP
- DNS

Care dintre cele de mai jos nu reprezintă o tehnologie relevantă pentru serviciul web:

- URL
- HTTP
- HTML
- SMB

Care dintre următoarele este o aplicație server SMTP?

- Postfix
- Courier
- Outlook
- Apache

Într-un client web (broswer) este tastată adresa:

```
ftp://bestman:none@test.com/lista/admisi.txt
```

Care dintre următoarele afirmații este adevărată?

- va fi inițiată o conexiune pe portul 80 către serverul test.com
- va fi descărcat fișierul /test.com/lista/admisi.txt
- se va folosi pentru autentificare parola "bestman"
- pentru stabilirea conexiunii utilizatorul "bestman" să trebuie să existe pe test.com

Pentru conectarea într-o rețea de partajare de fișiere folosind protocolul Direct Connect se poate face folosind Mozilla Firefox. Protocolul Bittorrent poate fi folosit atât pentru partajarea fișierelor ASCII, cât și pentru cele binare.

- adevărat, adevărat
- adevărat, fals
- fals, adevărat
- fals, fals

Care dintre afirmații este adevărată după lansarea comenzii:

```
wget --convert-links -r -l1 http://www.roedu.net/ -o sait001 &
```

- siteul www.roedu.net va putea fi navigat integral offline
- siteul va fi salvat local în directorul www.roedu.net
- în cazul în care siteul www.roedu.net va solicita autentificare, va fi trimis numele utilizatorului și parola acestuia
- utilizatorul va putea lansa în execuție alte aplicații înainte de terminarea descărcării siteului www.roedu.net

Care dintre următoarele aplicații nu poate fi folosită pentru conectarea la un server web:

- telnet
- ssh
- wget
- curl

Comunicația între două servere de mail se poate realiza folosind:

- POP3
- HTTP
- SMB
- nici una dintre variante

9 Elemente de securitate şi monitorizare

Social engineering bypasses all technologies, including firewalls.
Kevin Mitnick

Ce se învaţă din acest capitol?

- Problematica securităţii IT
- Elementele unei politici de securitate
- Securitatea sistemului de fişiere
- Securizarea accesului
- Monitorizarea locală a sistemului
- Tipuri de atacuri pentru reţelele de calculatoare
- Tipuri de viruşi
- Interceptarea traficului din reţea
- Scanarea de porturi

9.1 Problematica securităţii

De-a lungul ultimilor ani, multe teme tehnice specifice domeniului securităţii au părăsit domeniul IT, fiind preluate de ziare, jurnale TV, sau chiar de industria cinematografică. Din păcate procesul nu a fost, cel mai adesea, unul ce încerca aducerea în sfera publică a conceptelor de securitate, ci mai ales unul ce speculează senzaţionalul prin ignorarea constrângerilor lumii reale, ducând la promovarea unor noi mituri ale erei IT.

Deja am început să considerăm normale „performanţele" hackerilor din filmele americane, care reuşesc să compromită securitatea unui sistem în câteva secunde, şi sunt oameni ce cred că este posibil să scrii viruşi pentru sisteme de operare extraterestre (vezi „Ziua Independenţei", 1997).

Îi vom dezamăgi pe cei ce au deschis acest capitol în speranţa obţinerii unei puteri nemărginite în controlul tuturor sistemelor electronice, afirmând că cele 20 de pagini ce urmează nu reuşesc decât să aducă un pic de ordine în domeniul populat de mituri al securităţii IT. A deveni cu adevărat un profesionist în securitate este o carieră, după cum spunea şi Eric S. Raymond: *„Being able to break security doesn't make you a hacker anymore than being able to hotwire cars makes you an automotive engineer."*

9.1.1 Principii de bază

O primă clasificare a riscurilor de securitate distinge între trei tipuri de atacuri: atacuri venite din Internet (cu o rată de succes redusă), atacuri iniţiate din reţeaua locală, şi atacuri generate de pe aceeaşi maşină, acestea din urmă având un impact mult mai însemnat decât primele.

Deşi cu gradul de risc cel mai ridicat, atacurile iniţiate de utilizatorii serverului ţintă sunt deseori tratate în grabă şi unitar. Ne propunem în continuare discutarea aspectelor importante ce pot înlesni un astfel de atac, fiecare dintre acestea putând duce la compromiterea securităţii sistemului.

Principiul fundamental al securităţii, fie ea IT sau de orice altă natură, este:

Securitatea unui sistem este egală cu securitatea celei mai slabe verigi.

Altfel spus, degeaba îţi pui uşă ultra-performantă dacă nu o închizi cu cheia, sau aplicat în domeniul IT: nu are rost să cheltui sume enorme de bani pe sisteme de securitate, dacă utilizatorii folosesc drept parole propriul nume, sau îşi ţin parola lipită pe monitor.

Privit din perspectiva unui sistem IT, o soluţie de securitate trebuie să includă atât o politică de securitate, ce defineşte drepturile şi responsabilităţile utilizatorilor, cât şi specificaţii ale asigurării securităţii fizice, ale componentelor sistemului de operare, ale aplicaţiilor locale, precum şi a serviciilor de reţea.

O politică de securitate trebuie să stabilească un compromis între gradul de flexibilitate a serviciilor IT şi nivelul de securitate dorit. Luate *ad literam*, cerinţele de securitate ar presupune izolarea totală a sistemului de lumea exterioară, dar cum o astfel de abordare duce la limitarea funcţionalităţii, cel mai adesea securitatea unui sistem este definită ca un set de metode de protecţie menite să descurajeze şi să întârzie atacatorul.

9.2 Securizarea sistemului

9.2.1 Mecanisme de asigurare a securităţii în sistemul de operare

Securitatea aplicaţiilor trebuie să pornească de la asumarea gradului de risc dat de sistemul de operare. Dintre componentele sistemului de operare, o mare parte a atacurilor încearcă să exploateze limitări ale implementărilor de separare a memoriei, sau a sistemului de fişiere.

Un sistem de operare este sigur dacă resursele acestuia (zone de memorie, dispozitive de intrare/ieşire, fişiere, procese, etc.) sunt accesate în mod valid. Accesul valid este asigurat de nucleul sistemului de operare, ce acţionează ca un intermediar între utilizatori şi componentele hardware.

Pornind de la sistemele de operare simple (cele pentru PDA) până la sistemele mult mai complexe ce rulează pe calculatoarele personale, securitatea oferită de nucleu se bazează pe suport hardware. Procesoarele oferă cel puţin două niveluri de privilegiu: unul pentru operaţii obişnuite şi un altul pentru accesul la instrucţiuni privilegiate. Accesul la aceste instrucţiuni privilegiate este permis doar nucleului.

9.2.1.1 Securitatea memoriei

Problemele de securitate la nivelul memoriei ţin de detectarea corectă a încercărilor de accesare a unor zone de memorie ce nu se află în spaţiul de adrese al procesului respectiv, dar şi a încercărilor de schimbare a unor zone de memorie în spaţiul propriu de adrese, pe parcursul rulării procesului.

Fiecare proces are propriul său spaţiu de memorie virtuală mapat peste un spaţiu de memorie fizică printr-o tabelă de translatare ce aparţine procesului. Pentru un proces ce încearcă să acceseze o adresă de memorie pentru care nu există o translatare către o adresă fizică (adică pentru care nu există pagina respectivă de memorie în RAM), procesorul va verifica la nivel hardware dacă respectiva adresă se află în spaţiul de adrese al procesului. Dacă se află, înseamnă că respectiva pagină de memorie a fost evacuată pe disc (swap) şi trebuie adusă în RAM. În caz contrar va fi generat un semnal de pagină (de memorie) invalidă, pe baza căruia nucleul va putea decide suspendarea sau terminarea procesului în cauză. În sistemele

Unix semnalul se numeşte SIGSEGV, şi are ca efect terminarea procesului cu mesajul: „*Segmentation fault*".

Toate aceste operaţiuni cad în sarcina procesorului şi sunt realizate la nivel hardware.

Protecţia memoriei este o componentă esenţială a sistemelor multitasking (sisteme ce oferă posibilitatea mai multor procese de a se afla în stare de execuţie în acelaşi timp). Din punct de vedere hardware, primele procesoare pentru calculatoarele personale ce oferă suport pentru protecţia memoriei sunt procesoarele din familia 386. Cu toate acestea sistemele de operare până la Windows 98 şi Windows ME nu reuşesc să ofere o protecţie completă a memoriei.

Protecţia memoriei nu se limitează la restricţionarea acceselor la propriul spaţiu de adrese, ci se ocupă şi cu prevenirea inserării de cod executabil în diverse segmente ale unui proces. Pentru a detecta astfel de inserări, metoda cea mai folosită constă în marcarea zonelor executabile. Zonele marcate nu pot fi alterate pe parcursul execuţiei programului. Numele metodei este **W^X** (*Write xor eXecute*) indicând că o zonă de memorie poate fi ori executabilă, ori modificabilă, dar nu ambele.

Procesoarele din familia x86_64 oferă suport hardware pentru metoda W^X prin folosirea unui indicator NX (*Not eXecutable*) pentru marcarea zonelor ce pot fi modificate. Dacă o zonă marcată cu NX este executată nucleul este automat alertat.

9.2.2 Securizarea accesului

9.2.2.1 Securizarea fizică

Securitatea fizică include o gamă largă de parametri, de la controlul accesului până la asigurarea diferiţilor parametri optimi de funcţionare.

O primă componentă a securităţii fizice presupune plasarea echipamentelor în locaţii în care se poate asigura **controlul accesului**, pentru a preveni utilizarea neautorizată, distrugerea sau furtul.

Specificaţiile de **alimentare cu tensiune** vor include evaluări ale consumului de putere pentru dimensionarea surselor neinteruptibile de putere (UPS-uri), sau chiar a generatoarelor de tensiune.

Asigurarea temperaturii optime în centrele de operaţii ţine de folosirea unor sisteme de aer condiţionat sau de încălzire cu un grad ridicat de redundanţă.

Există şi cerinţe de **protecţie împotriva incendiilor**: o cameră cu servere va trebui să aibă, pe lângă sisteme de detecţie şi stingere a incendiilor, şi podea şi pereţi din materiale rezistente la foc.

O bună politică de securizare va urmări plasarea echipamentelor în bunkere aflate la subsolul clădirilor, urmărindu-se astfel şi **protecţia împotriva cutremurelor**.

9.2.2.2 Asigurarea autentificării

Privită prin prisma atacurilor, securitatea este definită ca un set de măsuri dinamice aflate la îndemâna administratorului de sistem. În realitate, orice soluţie de securitate porneşte de la definirea unui cadru de acţiune pentru utilizatori şi administratorii de sistem. Pe baza acestei **politici de securitate** sunt definite mecanismele de prevenire a atacurilor.

Una dintre componentele esenţiale ale unei politici de securitate este securitatea la nivelul utilizatorilor. Securitatea utilizatorilor se bazează pe separarea resurselor fiecărui utilizator, precum şi pe protejarea lor prin autentificare.

Un principiu important de prevenire a atacurilor este cel al drepturilor minime:

Orice operaţiune trebuie executată dintr-un cont cu drepturi cât mai limitate.

Un administrator novice va folosi contul privilegiat inclusiv pentru efectuarea unor operaţiuni uzuale. Exploatarea unor ambiguităţi ale variabilelor de mediu poate duce în acest

caz la lansarea accidentală în execuție a unei aplicații malițioase lăsată într-un spațiu public (gen `/tmp`), lansare ce nu ar fi fost posibilă dintr-un cont de utilizator.

În plus, lipsa unor servicii este cel mai adesea sesizată de către utilizator şi raportată administratorului, în vreme ce un set extins de drepturi trece mai degrabă neobservat de către utilizator.

Alegerea parolei

Deşi mecanismele de autentificare s-au diversificat, în continuare o pondere semnificativă o constituie perechile <utilizator, parolă> generate static, pentru o perioadă limitată de timp. O soluție alternativă, ce oferă un grad mult mai ridicat de scalabilitate, precum şi de securitate, este folosirea de **certificate**. Conceputul ce stă la baza folosirii de certificate se numeşte PKI (*Public Key Infrastructure*).

Numărul resurselor electronice fiind în continuă creştere, mulți utilizatori ajung să comită cel puțin una dintre erorile de căpătâi ale autentificării: alegerea de parole uşor de ghicit (spart), folosirea unei parole pentru mai multe conturi, sau pur şi simplu notarea parolelor în locuri nesigure.

Folosirea de parole prea simple. O parolă eficientă trebuie:

- să aibă un număr minim de caractere (se recomandă 7-8),
- să folosească atât minuscule, cât şi majuscule,
- să includă cel puțin un caracter special (şi nu doar pe ultima poziție în cadrul parolei),
- nu trebuie să fie un cuvânt din dicționar,
- nu trebuie să fie nici un nume de persoană.

Cu toate acestea, parolele trebuie să fie uşor de reținut, cea mai la îndemână soluție fiind alegerea unei fraze pe baza căreia se poate deduce parola. „*Come to the dark side, we have cookies!*" poate fi folosită pentru a reține parola: „Cttds, whc!", după cum „*Social engineering bypasses all technologies, including firewalls*" poate genera: „S3bat,1f".

În cazul folosirii unei parole ce nu respectă recomandările de mai sus, atacurile bazate pe dicționar (încercarea parolelor ce aparțin unui set de cuvinte comune), sau cele bazate pe forță (încercarea tuturor combinațiilor de caractere) pot avea rezultate aproape imediate. Există numeroase aplicații ce simulează astfel de atacuri, printre care şi *John the Ripper* (http://openwall.com/john/)

Folosirea aceleiaşi parole pentru conturi multiple. De la conturi triviale de download, până la conturile de email, sau la conturile bancare electronice, tot mai multe resurse solicită o autentificare pe bază de parolă. Ideal este să putem separa toate aceste conturi prin parole individuale. Cum acest lucru presupune memorarea unui număr ridicat de fraze, o soluție de compromis constă în folosirea unor parole comune pentru servicii similare cu miză de securitate redusă. Astfel, nu este prudent să folosim PIN-ul cardului de bancă pentru contul de mail, deoarece un atac de forță brută ar dura aproximativ o microsecundă. Putem însă folosi aceeaşi parolă pentru mai multe siteuri de descărcare de software.

Notarea parolelor. Deşi folosirea unor parole comune pentru servicii electronice cu miză mică reduce numărul parolelor, acesta rămâne ridicat. În plus, politicile de securitate solicită schimbarea periodică a parolelor: timpul de viață recomandat pentru o parolă este de şase luni. O soluție pentru parolele rar folosite constă în păstrarea lor centralizată. Există numeroase aplicații menite să protejeze printr-o singură parolă o bază de date de parole. Nu trebuie pierdut din vedere că în general aplicațiile moştenesc riscurile de securitate ale sistemului de operare pe care rulează, precum şi riscurile de securitate fizică ale sistemului pe care sunt stocate (fie el calculator personal, PDA sau server în Internet). Astfel recomandarea generală este să păstrați în aplicațiile de stocare a parolelor doar pe acelea asociate unor resurse marginal importante, şi să vă străduiți să memorați parolele pentru conturile bancare electronice.

Conştientizarea importanţei unei parole nu ţine doar de protejarea unor resurse personale, ci de gradul de securitate a întregului sistem, precum şi a reţelei locale. Altfel spus, într-o reţea în care utilizatorii folosesc parole triviale, riscul nu este doar unul personal. Un atacator poate să compromită din Internet un cont local, transformând un atac de la distanţă într-unul generat de pe aceeaşi maşină, ceea ce îi creşte mult şansele de succes. Acesta este şi motivul pentru care au evoluat şi mecanismele prin care administratorii de reţea pot forţa folosirea unor parole eficiente.

Atât reţelele Windows, cât şi cele Linux oferă suport pentru securitatea bazată pe **jetoane hardware** ce îşi regenerează parola la fiecare 30 sau 60 de secunde. Odată sincronizat ceasul serverului cu ceasul jetonului hardware, utilizatorul va folosi numele său de cont şi parola afişată de jeton la momentul autentificării. Preţul unei astfel de soluţii este de 5-10 euro per jeton hardware, fiecare jeton trebuind reconfigurat la 6 luni de zile. Trebuie remarcat totuşi că această soluţie nu rezolvă riscurile de securitate fizică a parolei.

Tipuri de parole

În debutul acestui capitol precizam că gradul de securitate a unui sistem este dat de securitatea celei mai slabe verigi. Un risc însemnat îl reprezintă complexitatea redusă a parolelor folosite de utilizatori sau chiar de administratorii de sistem.

Soluţiile de autentificare folosite la intrarea în sistemul de operare sunt doar o componentă a securităţii sistemului. După cum este prezentat şi în studiul de caz „Recuperarea parolei", absenţa parolei de BIOS corelată cu accesul fizic permit schimbarea parolei contului privilegiat. Pe lângă parolă de BIOS un sistem trebuie să protejeze editarea bootloader-ului, prin setarea unei parole.

Pentru setarea unei **parole pe grub** trebuie adăugat la începutul fişierului `/boot/grub/menu.lst`

```
timeout 3
password s3bat,1f
```

Prima linie specifică numărul de secunde după care va fi bootată imaginea implicită, în vreme ce a doua linie precizează parola necesară pentru a edita orice linie din grub.

Pentru a îmbunătăţi securitatea parolei de grub se poate opta pentru precizarea acesteia sub forma unui rezumat md5. Astfel, un atacator nu va putea afla parola chiar dacă ajunge să aibă acces de citire la fişierul `/boot/grub/menu.lst`:

```
password -md5 $1$qdbd2QUX$b3hUNT6vBUArya0mI5WI1
```

Utilitare pentru gestionarea parolelor

În sistemele Unix informaţiile specifice utilizatorilor sunt păstrate în fişierul `/etc/passwd`. Orice utilizator are acces de citire pentru acest fişier, din acest motiv s-a decis separarea informaţiilor legate de parole într-un nou fişier: `/etc/shadow`. Pentru acest fişier au drepturi de citire numai utilizatorul privilegiat şi grupul `shadow`. În plus, la instalarea implicită parolele sunt păstrate în format md5.

Utilitarul pentru schimbarea parolei este `passwd`. Aplicaţia nu permite rularea dintr-un script, astfel încât pentru generarea automată a parolelor se poate folosi utilitarul `pwgen`.

```
kiwi:~# pwgen -n -c 7 3
ahngeZ5 aiv5Iec bie3poR
```

Parametrul `-n` precizează în parolă va fi inclusă cel puţin o cifră, parametru `-c` de va defini lungimea parolei dorite. Ultima cifră reprezintă numărul de parole ce trebuie generate. Comanda de mai sus va genera trei parole a câte şapte caractere incluzând şi cel puţin o cifră.

9.2.2.3 Securitatea sistemului de fişiere

Securitatea sistemului de fişiere urmăreşte asigurarea izolării utilizatorilor prin separarea totală a fişierelor personale, sau doar prin definirea acţiunilor permise de către alţi utilizatori.

Fiecare utilizator dispune de o intrare (un director) în sistemul de fişiere pentru care are drepturi depline. Utilizatorul poate controla acţiunile permise pe fişierele conţinute în respectivul director. Unele intrări în sistemul de fişiere nu pot fi folosite decât de utilizatorul privilegiat.

Abordarea diferă între sistemele Windows şi cele Unix.

Sistemul de fişiere NTFS oferă de la versiunea 4.0 posibilitatea definirii unei liste de acces pentru fiecare utilizator din sistem. Această listă de acces este definită în raport cu şapte tipuri de acţiuni: *Full Control, Modify, Read & Execute, List, Read, Write, Special Permisions.* Subiectul drepturilor în NTFS este tratat mai pe larg în studiul de caz al acestui capitol.

În Unix definirea drepturilor se face pe trei trepte de privilegiu: proprietarul, membrii grupului din care face parte proprietarul, şi restul utilizatorilor. Observăm că nu sunt definite explicit drepturile utilizatorului privilegiat, acesta având aceleaşi drepturi ca şi proprietarul respectivului fişier.

Pentru fiecare dintre cele trei trepte de privilegiu sunt definite câte trei drepturi: citire, scriere şi execuţie.

- **Dreptul de citire** permite vizualizarea conţinutului unui director sau a unui fişier.
- **Dreptul de scriere** permite alterarea conţinutului unui fişier sau director. Pentru ştergerea unui fişier nu este necesar dreptul de scriere pe respectivul fişier, dar este necesar dreptul de scriere pe directorul din care acesta face parte.
- **Dreptul de execuţie** pentru un director înseamnă că este posibilă comutarea în respectivul director, iar pentru un fişier că se poate lansa în execuţie respectivul fişier.

În administrarea securităţii sistemului de fişiere în Unix, utilitarele de căpătâi sunt `chown` şi `chmod`. Primul oferă posibilitatea schimbării proprietarului şi a grupului căruia îi aparţine respectivul fişier, în vreme ce al doilea permite modificarea drepturilor.

```
kiwi:~# chown -R george:users test/
```

În exemplul de mai sus a fost efectuată o schimbare recursivă pentru toate fişierele şi subdirectoarele directorului test al proprietarului. Întreg conţinutul directorului test va aparţine utilizatorului `george` şi grupului `users`.

```
kiwi:~# ls -l hello.c
-rw-r--r--  1 razvan new 81 Oct  6 21:35 hello.c
```

Este listat fişierul `hello.c` din directorul curent. Primele 10 caractere definesc tipul intrării în sistemul de fişiere şi a drepturilor asociate:

- "`-`" este folosit pentru identificarea de fişiere ("`d`" pentru directoare)
- caracterele 2, 3, 4 definesc drepturile proprietarului în ordinea: citire, scriere, execuţie. "`rw-`" înseamnă că proprietarul are drept de citire şi scriere, dar nu şi de execuţie
- caracterele 5, 6, 7 definesc drepturile grupului. În cazul exemplului de mai sus toţi utilizatorii din grupul `new` vor avea doar drept de citire.
- Caracterele 8, 9, 10 definesc drepturile celorlalţi utilizatori din sistem.

Sintaxa utilitarului `chmod` permite atât exprimarea drepturilor prin expresii regulate, cât şi în format octal.

Pentru schimbarea drepturilor vom folosi mai întâi **expresii regulate**. Drepturile pot fi precizate pentru oricare dintre cele trei niveluri de privilegiu: utilizator, grup, restul utilizatorilor printr-o singura literă: `u`, `g`, `o`. Drepturile pot fi

- adăugate prin folosirea operatorului "`+`"

- înlăturate prin folosirea operatorului "-"
- precizate explicit prin folosirea operatorului "="

```
kiwi:~# chmod o+x hello.c
kiwi:~# ls -l hello.c
-rw-r--r-x  1 razvan new 81 Oct  6 21:35 hello.c
kiwi:~# chmod u=rx hello.c
kiwi:~# ls -l hello.c
-r-xr--r-x  1 razvan new 81 Oct  6 21:35 hello.c
```

În exemplul de mai sus am adăugat pentru restul utilizatorilor dreptul de execuţie mai întâi, iar la al doilea pas am precizat explicit că drepturile proprietarului trebuie să fie "rx"adică doar citire şi execuţie.

Pentru a schimba drepturile folosind **exprimarea octală** trebuie privită codificarea fiecărui grup de drepturi ca un număr binar exprimat pe trei biţi. Astfel "rw-" va fi interpretat ca numărul 110, adică tradus în octal 6.

Modificarea se va face prin precizarea explicită a drepturilor pentru toate cele trei niveluri de privilegiu, cu alte cuvinte prin precizarea unui număr de trei cifre.

Pentru a acorda toate drepturile proprietarului fişierului, drepturi de citire şi execuţie grupului, şi doar drepturi de citire pentru restul utilizatorilor este suficientă o singură comandă:

```
kiwi:~# chmod 754 hello.c
kiwi:~# ls -l hello.c
-rwxr-xr--  1 razvan new 81 Oct  6 21:35 hello.c
```

Pentru controlul drepturilor unei noi intrări în sistemul de fişiere se foloseşte un parametru de restricţie numit **umask**. Drepturile efective ce vor fi atribuite unei noi intrări în sistemul de fişiere se obţin prin efectuarea operaţiei de ŞI logic între valoarea inversată a umask şi permisiunile implicite (666 pentru fişiere şi 777 pentru directoare).

Utilitarul ce permite inspectarea şi modificarea parametrului umask se numeşte `umask`.

```
kiwi:~# umask
0022
kiwi:~# touch uso7_test1
kiwi:~# ls -l uso7_test1
-rw-r--r--  1 razvan razvan 0 Nov 10 17:28 uso7_test1
kiwi:~# umask 027
kiwi:~# touch uso7_test2
kiwi:~# ls -l uso7_test2
-rw-r-----  1 razvan razvan 0 Nov 10 17:28 uso7_test2
```

9.2.3 Limitarea şi extinderea privilegiilor

Sistemele Unix oferă suport în principal doar pentru două niveluri de privilegiu: utilizatori neprivilegiaţi şi administrator (root). În practică este necesar un grad mai ridicat de flexibilitate în acordarea privilegiilor, pentru a permite rularea unor aplicaţii privilegiate de către utilizatorii obişnuiţi. Acest lucru se realizează prin limitarea sau extinderea privilegiilor la nivel de aplicaţie, sau chiar la nivel de utilizator.

9.2.3.1 Folosirea opţiunilor de suid

O categorie aparte de aplicaţii o reprezintă utilitarele ce necesită resurse speciale din nucleu. Pentru a putea da posibilitatea şi utilizatorilor neprivilegiaţi de a rula astfel de programe, pe parcursul execuţiei identificatorul utilizatorului (uid) se schimbă, astfel încât respectivele secţiuni sunt rulate drept root. La încheierea unei astfel de secţiuni se revine la identificatorul utilizatorului ce a lansat aplicaţia.

Marcarea unei astfel de aplicații se face folosind un bit special denumit **suid** (*switch user id*). Această metodă permite execuția unui program cu drepturile celui ce îl deține (de obicei utilizatorul privilegiat).

Pentru controlul bitului de suid se folosește utilitarul `chmod`:

```
kiwi:~# ls -l a.out
-rwxr-xr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
kiwi:~# chmod 2755 a.out
kiwi:~# ls -l a.out
-rwxr-sr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
kiwi:~# chmod 4755 a.out
kiwi:~# ls -l a.out
-rwsr-xr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
```

Trebuie totuși avut în vedere că marcarea unui executabil cu dreptul de suid reprezintă un potențial risc de securitate. Dacă pe parcursul execuției utilizatorul reușește să forțeze terminarea anormală a programului înainte de refacerea uid-ului, poate obține acces la un shell de root. Din acest motiv un pas important în securizarea unui sistem este determinarea tuturor executabilelor ce au setat bitul de suid, și desuidarea aplicațiilor ce nu sunt folosite. Pentru aceasta se va folosi comanda `find`:

```
kiwi:~# find / -perm -4000 -o -perm -2000 -type f -print
```

Printre utilitarele ce au nevoie de suidare pentru a putea fi rulate de către utilizatori obișnuiți se află `ping` și `traceroute`. Pe lângă acestea, un sistem Linux nou instalat va mai avea câteva zeci de programe suidate.

9.2.3.2 Schimbarea rădăcinii sistemului de fișiere (chroot)

Unul dintre mecanismele extrem de eficiente în securizarea unor servicii, sau pur și simplu în izolarea totală a utilizatorilor, se numește `chroot`. Acesta presupune virtualizarea sistemului de fișiere pentru respectivul daemon sau utilizator.

Acest mecanism reduce riscurile de securitate. Dacă un atacator obține acces cu ajutorul unei vulnerabilități într-un proces ce folosește `chroot`, el va avea acces limitat la sistemul de fișiere. Pe de altă parte, folosirea acestui mecanism este destul de dificilă, datorită faptului că în general programele sunt compilate pentru a folosi biblioteci. Din această cauză, bibliotecile trebuie incluse în directorul văzut de proces ca rădăcină. O altă soluție este compilarea statică a programelor, fără biblioteci.

Să luam de exemplu daemonul de DNS: bind9. Dacă modificăm scriptul de inițializare adăugând opțiunea `-t /var/dns`, serviciul va porni considerând ca rădăcină a întregului sistem de fișiere `/var/dns/`. Primul fișier căutat va fi `/etc/bind/named.conf`, astfel că înainte de lansarea serviciului va trebui copiat `/etc/bind/named.conf` în `/var/dns/etc/bind/named.conf`. Pe lângă acest fișier vor trebui copiate fișierele de zonă (necesare configurării serviciului de DNS), fișierele de jurnalizare, dar și unele fișiere speciale gen `/dev/null` sau `/dev/log`.

Majoritatea programelor ce implementează serviciile UNIX oferă posibilitatea rulării procesului cu sistemul de fișiere limitat. Dacă programul nu oferă această posibilitate, sau dacă se dorește testarea mecanismului, se poate folosi utilitarul `chroot`.

Utilitarul chroot

În exemplul de mai jos se rulează utilitarul `chroot` într-un nou director `/root/test`. Dacă nu se specifică ce comandă se rulează, implicit se execută interpretorul de comenzi precizat de variabila de mediu `SHELL`. Cum `/bin/sh` este o legătură către `/bin/bash`, este necesară copierea sa în viitorul sistem de fișiere. Cu toate acestea la rularea comenzii `bash` prin `chroot`

nu poate fi executată. Motivul este lipsa bibliotecilor. Se afişează apoi lista bibliotecilor necesare `bash`, iar respectivele biblioteci sunt apoi copiate în viitorul sistem de fişiere.

Comanda `chroot` poate fi acum executată, rezultând un prompt într-un sistem de fişiere ce are doar 2 directoare şi 4 fişiere. Se pot totuşi rula comenzile interne `bash`.

```
kiwi:~# pwd
/root
kiwi:~# mkdir test
kiwi:~# mkdir test/bin
kiwi:~# cp /bin/bash test/bin/
kiwi:~# chroot /root/test
chroot: cannot run command `/bin/bash': No such file or directory
kiwi:~# ldd /bin/bash
                libncurses.so.5 => /lib/libncurses.so.5 (0x4001c000)
        libdl.so.2 => /lib/tls/libdl.so.2 (0x4005b000)
        libc.so.6 => /lib/tls/libc.so.6 (0x4005f000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
kiwi:~# mkdir test/lib
kiwi:~# cp /lib/libncurses.so.5 /lib/libdl.so.2 test/lib
kiwi:~# cp /lib/libc.so.6 test/lib
kiwi:~# chroot /root/test
kiwi:/# pwd
/
```

9.2.3.3 sudo

Utilitarul `sudo` permite utilizatorilor neprivilegiaţi să ruleze unele comenzi având identitatea (uid-ul) altui utilizator (în general a utilizatorului root). Controlul utilizatorilor ce au dreptul să ruleze comenzi privilegiate, precum şi lista exactă a comenzilor este păstrată în fişierul `/etc/sudoers`. Pentru a putea funcţiona sudo, fişierul `/etc/sudoers` trebuie să aibă doar drepturi de citire pentru proprietar şi grup, altfel veţi obţine mesajul:

```
sudo: /etc/sudoers is mode 0640, should be 0440
```

Pentru editarea `/etc/sudoers` este recomandat utilitarul `visudo`, evitând astfel oferirea unor drepturi provizorii de scriere şi minimizând de asemenea şi alte riscuri de securitate.

La lansarea unei comenzi `sudo`, după verificarea fişierului `/etc/sudoers` va trebui introdusă parola utilizatorului ce a lansat comanda `sudo` (nu a contului privilegiat).

Pentru exemplul următor să presupunem că utilitarul `/bin/ping` a fost desuidat. Pentru a permite utilizatorului `rookie` să folosească comanda `ping`, administratorul va trebui să adauge în fişierul `/etc/sudoers` următoarea linie:

```
rookie ALL=(ALL) /bin/ping
```

Acum din contul utilizatorului `rookie` vom încerca mai întâi să rulăm un interpretor de comenzi, iar apoi comanda `ping`:

```
rookie@kiwi:~$ sudo /bin/bash
Sorry, user rookie is not allowed to execute '/bin/bash' as root on
localhost.
rookie@kiwi:~$ /bin/ping 141.85.99.1
-su: /bin/ping: Permission denied
rookie@kiwi:~$ sudo /bin/ping 141.85.99.1
PING 141.85.37.1 (141.85.37.1): 56 data bytes
64 bytes from 141.85.37.1: icmp_seq=0 ttl=64 time=0.5 ms
```

Rularea unei comenzi `sudo` fără a avea o corespondenţă în fişierul `/etc/sudoers` va fi jurnalizat ca un incident de securitate. Pentru a verifica lista comenzilor ce pot fi rulate ca root se poate folosi opţiunea `-l` a comenzii `sudo`:

```
rookie@kiwi:~$ sudo -l
User rookie may run the following commands on this host:
    (ALL) /bin/ping
```

9.2.3.4 Limitarea numărului de procese şi fişiere

Utilizarea ulimit

Utilitarul `ulimit` permite controlul resurselor alocate pentru un proces pornit din interpretorul de comenzi `bash`. În realitate ulimit este o comandă internă a `bash`.

Pentru inspectarea limitărilor curente se poate folosi parametrul `-a`. Acest parametru ne va oferi inclusiv parametrii pentru configurarea diferitelor limite disponibile prin `ulimit`.

```
kiwi:~# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) unlimited
virtual memory          (kbytes, -v) unlimited
kiwi:~# ulimit -n 2048
kiwi:~# ulimit -n
2048
```

În exemplu de mai sus a fost realizată o listare a tuturor limitărilor curente, pentru ca apoi să fie redefinit numărul maxim de fişiere deschise la 2048.

În scopul prevenirii unui atac DoS local (cauzat de crearea unui număr de procese extrem de mare) se recomandă limitarea numărului de procese pentru utilizatorul privilegiat prin adăugarea în fişierul `/root/.bashrc` a unei limite de procese:

```
ulimit -u 1024
```

Utilizarea /etc/security/limits.conf

Fişierul `/etc/security/limits.conf` reprezintă un mecanism eficient pentru limitarea resurselor disponibile unui utilizator sau unui grup de utilizatori. Acest mecanism este o componentă a pachetului PAM, astfel că limitările precizate se vor aplica doar programelor ce utilizează PAM. Pachetul PAM (*Pluggable Authentication Modules*) conţine o serie de biblioteci care se ocupa de autentificarea aplicaţiilor sau daemonilor.

Se pot defini două tipuri de limite, denumite limite software şi limite hardware. La depăşirea unei limite software se generează un mesaj de atenţionare. La depăşirea unei limitări definite ca hardware procesul respectiv este terminat fără nici o altă atenţionare.

Sintaxa fişierului se bazează pe definirea de reguli.

O regulă are patru elemente:

- **ţinta**: utilizatorii sau grupurile pentru care se aplică respectiva regulă;
- **tipul de regulă**: regulile pot fi definite ca software sau hardware;
- **tipul de limitare**: este precizat parametrul care este limitat;
- **limitarea**: valoarea la care se realizează limitarea.

Pentru câmpul ţintă precizarea unui grup se face prin precedarea numelui de grup cu caracterul `"@"`.

De exemplu, ne propunem să limităm numărul de procese a utilizatorului rookie la 35, iar la depăşirea limitei de 20 de procese să fie generat un mesaj de avertisment. În plus dorim să

limităm numărul de autentificări permise utilizatorului rookie la 10. Pentru aceasta vom adăuga în fişierul `/etc/security/limits.conf` următoarele trei linii:

```
rookie soft nproc 20
rookie hard nproc 35
rookie -    maxlogins 10
```

Tot în pachetul PAM avem la dispoziţie un mecanism similar accesibil prin editarea fişierului `/etc/limits`. Sintaxa acestui fişier diferă de cea a `/etc/security/limits.conf`. În plus, regulile nu pot fi definite pentru grupuri, ci doar pentru utilizatori.

Pentru a descrie limitările folosite în exemplul anterior vor trebui adăugate în fişierul `/etc/limits` următoarele linii:

```
rookie U 35
rookie L 10
```

9.2.3.5 Limitarea spaţiului ocupat pe disc

Utilitarul `quota` este folosit pentru a limita spaţiul ocupat de fiecare utilizator pe disc. Prezentarea instalării şi configurării acestui utilitar este realizată în secţiunea de studii de caz. (vezi 9.6.3)

9.3 Întreţinerea sistemului

9.3.1 Monitorizarea locală în timp real

9.3.1.1 Monitorizarea încărcării sistemului

Monitorizarea dinamică (în timp real) a sistemului urmăreşte nivelul de încărcare a diferitelor resurse hardware. Astfel, pentru a determina gradul de încărcare a procesorului se poate folosi comanda `uptime`. Pentru a urmări nivelul de folosire a memoriei RAM se poate folosi comanda `free`.

Comanda `top` combină informaţiile oferite de comenzile `uptime` şi `free`.

Vom analiza în continuare rezultatul rulării comenzii `top`:

```
top - 14:05:43 up  8:12,  2 users,  load average: 1.01, 1.02, 1.00
Tasks:  61 total,   3 running,  58 sleeping,   0 stopped,   0 zombie
Cpu(s): 99.7% us,  0.3% sy,  0.0% ni,  0.0% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:    256740k total,   254200k used,     2540k free,    12680k buffers
Swap:   289128k total,     1444k used,   287684k free,    41356k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2490 www-data  25   0 26204  18m  12m R 99.5  7.5 275:26.52 apache
14067 root      16   0  2064 1036 1852 R  0.3  0.4   0:00.02 top
    1 root      16   0  1504  512 1352 S  0.0  0.2   0:00.73 init
    2 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    3 root       5 -10     0    0    0 S  0.0  0.0   0:00.00 events/0
    4 root       8 -10     0    0    0 S  0.0  0.0   0:00.00 khelper
    5 root      14 -10     0    0    0 S  0.0  0.0   0:00.00 kacpid
   31 root       5 -10     0    0    0 S  0.0  0.0   0:03.15 kblockd/0
   41 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
```

Prima linie conţine (în acelaşi format) aceleaşi informaţii pe care le oferă comanda `uptime`: ora curentă, urmată de intervalul de timp de la ultima iniţializare a sistemului (în cazul exemplului 8 ore şi 12 minute), numărul de utilizatori autentificaţi în sistem, precum şi încărcarea medie a sistemului pentru ultimele 1, 5 sau 15 minute. O încărcare supraunitară este echivalentă cu un sistem solicitat.

Liniile 4 şi 5 oferă informaţiile specifice comenzii `free`. Astfel linia 4 descrie încărcarea memoriei RAM în vreme ce linia 5 indică nivelul de utilizare a partiţiei de swap. Folosirea unei

cantități însemnate de memorie din swap indică faptul că memoria RAM este insuficientă pentru cerințele sistemului.

În exemplul de mai sus se observă că procesorul este supraîncărcat, în vreme ce memoria RAM este bine dimensionată (pentru că memoria swap nu este folosită). Inspectând mai departe procesele cele mai relevante din punctul de vedere al consumului de resurse se observă că procesul `apache` a petrecut în procesor 275 de minute, adică mai mult de jumătate din timpul scurs de la ultima inițializare a sistemului. Acesta este un comportament neobișnuit. Pentru un sistem bine configurat cele mai active procese vor reuşi la nivelul unei luni de zile să ocupe 1-2 minute.

Concluzia analizei dinamice a sistemului de mai sus nu este că resursele hardware sunt insuficiente, ci că unul dintre servicii (mai exact serviciul de web) are un comportament anormal. Prima măsură ar trebui să fie oprirea procesului `apache` şi inspectarea configurațiilor specifice acestui daemon.

Comanda `top` este prezentată pe larg în capitolul de procese. [vezi 4.2.4]

9.3.1.2 Monitorizarea utilizării discului

O altă resursă importantă a unui sistem o reprezintă spațiul liber de pe hard-disk. Pentru aceasta trebuie monitorizat nivelul de ocupare a partițiilor definite pe disc, dar şi dimensiunea unor directore şi fişiere.

Spațiul liber pentru o partiție este obținut utilizând comanda `df`:

```
cursuri:/home/courses# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda1             4.6G  3.2G  1.2G  74% /
tmpfs                 126M  4.0K  126M   1% /dev/shm
/dev/hda6              32G   30G  1.2G  97% /home
```

În exemplu de mai sus spațiul pe disc este împărțit în două partiții: una folosită pentru rularea sistemului (montată ca "`/`") şi o a doua folosită pentru stocarea datelor utilizatorilor (montată ca "`/home`"). În acest fel, chiar dacă spațiul pe partiția `/home` este epuizat, acest lucru nu va afecta funcționarea sistemului.

Din rezultatul comenzii `df` se mai poate deduce şi prezența unui card de memorie cu o capacitate de 128 MB. Parametrul `-h` a fost folosit pentru a da o formă mai uşor de citit rezultatului comenzii.

În cazul în care se doreşte inspectarea dimensiunii unui fişier anume se poate folosi comanda `ls`:

```
cursuri:/var/log# ls -lh kern.log
-rw-r----- 1 root adm 62K Aug 10 14:21 kern.log
```

Pentru a obține dimensiunea unui director comanda utilizată este `du`:

```
cursuri:/home/courses# du -sh rc
155M    rc
```

9.3.1.3 Monitorizarea serviciilor de rețea

Pentru monitorizarea serviciilor de rețea disponibile pe stația locală este folosit utilitarul `netstat`.

Apelat fără nici un parametru, `netstat` va afişa o listă cu toate conexiunile active (porturile aflate în starea de *listen* se consideră inactive).

O utilizare frecventă a acestui utilitar urmăreşte testarea funcționalității anumitor servere (http, ftp, etc.). În general, aceste servere aşteaptă conexiuni TCP sau UDP pe anumite porturi.

Pentru a obține lista tuturor conexiunilor TCP, se poate folosi:

```
# netstat --tcp --all
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 dhcp-204.cs.pub.ro:33002 64.233.183.19:www ESTABLISHED
tcp6 0 0 *:ssh *:* LISTEN
tcp6 0 0 ip6-localhost:smtp *:* LISTEN
tcp6 0 0 ip6-localhost:953 *:* LISTEN
[..]
```

Pentru a verifica funcţionarea diferitelor servere instalate, care ascultă eventuale cereri de conectare, este utilă comanda:

```
# netstat --tcp --listening --numeric-ports
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 127.0.0.1:53 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:23 0.0.0.0:* LISTEN
tcp 0 0 127.0.0.1:25 0.0.0.0:* LISTEN
[..]
```

Se observă că pe maşina locală există servere care ascultă pe portul 23 (telnet), 25 (SMTP) sau 53 (DNS). Aşadar, server-ele telnet, SMPT şi DNS sunt funcţionale.

Dacă se doreşte vizualizarea conexiunilor deja stabilite, se poate folosi comanda:

```
# netstat --tcp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 dhcp-204.cs.pub.r:32776 cs23.msg.dcn.yahoo:5050 ESTABLISHED
tcp 0 0 dhcp-204.cs.pub.r:33002 64.233.183.19:www ESTABLISHED
```

Pentru vizualizarea conexiunilor UDP, se poate utiliza parametrul `"-u"` sau `"--udp"`.

Să presupunem că descoperim un server care funcţionează, el fiind în starea *listen* pe portul 22. Pentru stoparea acestui server putem opri procesul corespunzător. Înainte însă, trebuie aflat *pid-ul* acestui proces (opţiunea `-p`):

```
# netstat --tcp --listening --program --numeric-ports
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      5721/apache
[..]
tcp6       0      0 :::22                   :::*                    LISTEN      5515/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      5505/master
```

Ulterior, se poate trimite un semnal KILL către procesul cu numărul 5515.

```
# kill -KILL 5515
```

9.3.1.4 Monitorizarea activităţii utilizatorilor

Pentru a determina ce utilizatori sunt autentificaţi la un moment dat în sistem se foloseşte comanda `who`:

```
cursuri:~# who
so       pts/0        Aug 10 13:50 (anaconda.cs.pub.ro)
ot       pts/1        Aug 10 14:05 (kiwi.cs.pub.ro)
```

Din exemplul de mai sus se poate deduce că în sistem sunt autentificaţi utilizatorii so şi ot. Ambii utilizatori au acces în sistem prin emulatoare de terminal (pts), cu alte cuvinte prin intermediul unor sesiuni la distanţă. Sesiunea utilizatorului so a fost iniţiată la ora 13:50 de pe staţia anaconda.cs.pub.ro, în vreme ce sesiunea utilizatorului ot a fost începută la 14:05 de pe sistemul kiwi.cs.pub.ro.

9.3.2 Folosirea jurnalelor (auditarea sistemului)

9.3.2.1 Crearea jurnalelor

`Syslogd` este un utilitar care permite lucrul cu mesajele sistem şi cele generate de kernel. Pe scurt, aplicaţiile trimit diferite mesaje către `syslogd` iar acesta, utilizând un fişier de configurare, depune aceste mesaje în anumite fişiere, le scrie în anumite console sau eventual le trimite către o altă maşină.

În Debian utilitarul `syslogd` este disponibil în pachetul `sysklogd`.

```
# apt-get install sysklogd
```

Fişierul de configurare este `/etc/syslog.conf`. Acest fişier conţine reguli care definesc modurile în care sunt utilizate mesajele sistem primite.

O regulă este formată din două câmpuri: un câmp de selecţie prin care selectează anumite mesaje, şi un câmp de acţiune care specifică ce anume se va face cu mesajele selectate. Cele două câmpuri sunt separate prin spaţii sau tab-uri. Un exemplu simplu de regulă care scrie toate mesajele de eroare provenite de la sistemul de mail în fişierul `/var/log.mail.err` este:

```
mail.err                        /var/log/mail.err
```

Să presupunem că maşina *kiwi* doreşte trimirea tuturor mesajelor de la sistemul de mail către maşina *lorien*.

Pe kiwi adăugăm în `/etc/syslog.conf`:

```
mail.*                          @lorienlog
```

iar pe `lorienlog` îl definim în `/etc/hosts` astfel:

```
141.85.37.226   lorienlog
```

Pe lorien, implicit `syslogd` nu acceptă conexiuni de la alte maşini, dar adăugând opţiunea `-r` la rularea lui, acesta va accepta pe portul 514 conexiuni UDP. Pentru aceasta modificăm în `/etc/init.d/sysklogd` variabila `SYSLOGD`: `SYSLOGD= "-r"`

Pentru ca modificările din `/etc/syslog.conf` să aibă efect trebuie repornit `syslogd`:

```
# /etc/init.d/sysklogd restart
```

Deşi este foarte util pentru monitorizarea sistemului, *syslogd*-ul prezintă şi unele dezavantaje. De exemplu, o aplicaţie neglijentă ar putea genera foarte multe mesaje, ceea ce ar putea duce la umplerea discului şi la instabilitatea întregului sistem. Trimirea mesajelor către o altă maşină, mai puţin importantă în reţea, ar rezolva această problemă. Şi în acest caz însă, pe lângă faptul că se expune acea maşină la acelaşi risc, trebuie luate precauţii suplimentare precum limitarea staţiilor de la care se primesc mesaje pe portul 514 utilizând UDP.

9.3.2.2 Păstrarea jurnalelor

Pentru păstrarea centralizată a jurnalelor se foloseşte directorul `/var/log`. Numărul fişierelor din acest director depinde de configurarea daemonului de monitorizare syslogd, precum şi a diferitelor servicii. De exemplu, daemonul apache se va ocupa singur de actualizarea propriilor jurnale.

Orice intrare într-un fişier jurnal va avea o etichetă de timp, urmată de un mesaj, dar formatul acestor mesaje diferă în funcţie de tipul fişierului jurnal. Spre exemplu, jurnalele mesajelor de nucleu (`kern.log`) vor fi diferit formatate faţă de jurnalele de email (`mail.log`) sau de cele legate de activităţile daemonilor (`daemon.log`)

În exemplul de mai jos sunt redate câteva linii din jurnalul de acces al serviciului de web:

```
cursuri:/home/courses# less /var/log/apache/access.log
64.1.215.164 - - [05/Aug/2007:07:06:43 +0300] "GET /~ixlabs/lxr/ident?a=m68k;v=valgrind-
3.2.3;i=X86G_FC_MASK_C1 HTTP/1.0" 200 4668 "-" "Mozilla/5.0 (Twicel
```

```
er-0.9 http://www.cuill.com/twiceler/robot.html)" "-"
64.1.215.164 - - [05/Aug/2007:07:06:47 +0300] "GET /~ixlabs/lxr/ident?a=m68k;v=valgrind-
3.2.3;i=ELFCLASS32 HTTP/1.0" 200 5323 "-" "Mozilla/5.0 (Twiceler-0.
9 http://www.cuill.com/twiceler/robot.html)" "-"
64.1.215.164 - - [05/Aug/2007:07:06:53 +0300] "GET /~ixlabs/lxr/ident?a=sparc64;v=valgrind-
3.2.3;i=filename_base HTTP/1.0" 200 7112 "-" "Mozilla/5.0 (Twice
ler-0.9 http://www.cuill.com/twiceler/robot.html)" "-"
```

9.3.2.3 Inspectarea jurnalelor

Pentru citirea jurnalelor se poate folosi orice editor de text de la `vi` la `less`. Datorită faptului că mesajele recente sunt ataşate la sfârşitul fişierului jurnal, un utilitar des folosit este aplicaţia `tail`.

Pentru inspectarea dinamică a jurnalelor se foloseşte comanda `tail` cu opţiunea `-f`. Comanda `tail` afişează în mod implicit ultimele 10 linii dintr-un fişier dat. Prin folosirea parametrului `-f` se asigură afişarea dinamică a liniilor adăugate la respectivul fişier.

Un exemplu de folosire pentru inspectarea dinamică a activităţii daemonilor de mail (smtp, imap, etc) este prezentat mai jos:

```
cursuri:/var/log# tail -f /var/log/mail.log
```

9.3.2.4 Rotirea jurnalelor

Pentru a evita problema ocupării în totalitate a spaţiului de pe disc există un utilitar special, numit `logrotate` care permite reutilizarea fişierelor de jurnalizare zilnic, săptămânal sau când de exemplu au depăşit o anumită dimensiune. Principiul după care funcţionează este următorul: pentru fiecare fişier jurnal (`test.log`) se efectuează periodic rotaţii. Mai precis, se salvează conţinutul său într-un fişier de backup cu un nume format din numele original la care se adaugă un număr, copia având un număr mai mare cu cât este mai veche (`test.log.1`, `test.log.2` etc). La fiecare rulare, `logrotate` va şterge fişierul cel mai vechi şi le va actualiza pe celelalte corespunzător.

`Logrotate` este de obicei apelat zilnic din `cron` iar fişierul său de configurare este `/etc/logrotate.conf`.

De exemplu, dacă se doreşte ca fişierul `/var/log/test` să fie rescris săptămânal, păstrându-se alte 2 cópii mai vechi, `test.0` şi `test.1`:

```
/var/log/test {
        rotate 3
        weekly
}
```

9.3.2.5 Alte utilitare de jurnalizare

Utilitarul `dmesg` poate fi folosit pentru afişarea informaţiilor de la iniţializarea sistemului. Acestea sunt mesaje ale nucleului şi pot fi obţinute şi prin vizualizarea directă a jurnalelor de nucleu. Pentru configuraţiile implicite jurnalele de nucleu sunt păstrate în fişierul `/var/log/kern.log`.

Pentru monitorizarea activităţii utilizatorilor se poate folosi comanda `last`. Informaţiile referitoare la sesiunile de lucru sunt păstrate în format binar în fişierul `wtmp`.

```
cursuri:/var/log# last
ot       pts/1        kiwi.catc.ro     Fri Aug 10 14:27   still logged in
ot       pts/1        kiwi.catc.ro     Fri Aug 10 14:05 - 14:21  (00:16)
so       pts/0        anaconda.cs.pub. Fri Aug 10 13:50   still logged in
reboot system boot 2.6.8-2-686      Fri Aug 10 05:53          (08:54)
```

9.4 Riscuri de securitate în rețele de calculatoare

9.4.1 Tipuri de atacuri în rețea

Atacurile de rețea sunt clasificate în funcție de gravitate în trei categorii: atacuri de recunoaştere, atacuri DoS, şi atacuri de obținere a accesului.

Atacurile de recunoaştere includ aplicații de complexități diferite. Cel mai simplu atac de recunoaştere poate fi generat printr-un script ce inventariază stațiile dintr-o rețea folosind pachete de ping. Pentru a obține informații în legătură cu serviciile rulate pe o mașină se poate folosi o aplicație de scanat porturi (spre finalul acestui capitol este prezentat utilitarul `nmap`). Aplicațiile mai complexe de recunoaştere oferă nu numai lista serviciilor, dar chiar şi o listă a posibilelor vulnerabilități. Un astfel de utilitar este `nessus` (http://www.nessus.org/)

Atacurile DoS (*Denial of Service*) au ca scop congestionarea unui server sau doar a unui serviciu, în scopul reducerii funcționalității.

Se poate distinge între atacurile clasice DoS, bazate în general pe *flooding*, şi atacurile DoS distribuite, numite DDoS, ce au la bază un mecanism de propagare (un virus) pe un număr cât mai mare de destinații şi sincronizarea atacului DoS către o țintă.

Un *flood* reprezintă un număr foate mare de pachete venite într-un interval scurt de timp. Un trafic ce poate fi interpretat drept *flood* la nivelul ruterului de acces în rețeua locală poate fi tratat drept trafic normal în nucleul Internetului. Pentru tehnologiile actuale un *flood* este definit astfel: la nivelul unui switch de nivel 2 un număr de 100.000- 150.000 de pachete pe secundă, la nivelul unui ruter de acces (ce conectează o rețea locală) un *flood* are 8.000-12.000 de pachete pe secundă, iar în nucleul Internetului un trafic de milioane de pachete pe secundă poate fi tratat drept trafic legitim. În ciuda capacității de prelucrare a unui volum de pachete de ordinul zecilor de mii pe secundă, complexitatea adăugată de modele de *flood* specifice poate reduce limita de pachete pe secundă chiar la câteva sute.

Atacurile bazate pe *flooding* pot folosi conexiuni TCP, mai exact pachete de SYN (atacul numindu-se *SYN flooding*), dar şi pachete ICMP sau UDP. În cazul *SYN flooding* atacatorul va trimite un număr foarte mare de cereri de deschidere a unei noi conexiuni TCP (pachete ce au câmpul de control SYN setat). Serverul țintă va răspunde cu pachete ce au câmpurile ACK şi SYN setate, dar atacatorul nu va trimite niciodată pachetul de stabilire a conexiunii (o sesiune TCP se bazează pe *three-way handshake*). Astfel sesiunile vor rămâne în starea *half-open*, consumând resurse în continuare pe server.

Mecanismele de protecție împotriva atacurilor *SYN flood* se bazează pe stabilirea unui timp maxim pentru stabilirea unei conexiuni (timp în care conexiunea poate fi în starea *half-open*), sau a unui număr maxim de conexiuni *half-open* (valoare de la care conexiunile *half-open* vor începe să fie şterse).

ICMP flooding se bazează pe trimiterea unui număr foarte mare de pachete ICMP, consumând întreaga lățime de bandă disponibilă. Cel mai adesea atacul este prevenit prin filtrarea întreg traficului ICMP, cu costul pierderii funcționalități utilitarelor `ping` şi `traceroute`.

Atacul *UDP flooding* este cu adevărat distructiv când foloseşte drept țintă porturile 7 şi 19, adică serviciile de echo şi chargen. Aceste servicii fiind rar folosite, în rețele locale pot fi oprite de firewall-ul de intrare în LAN, fără un impact real asupra funcționării rețelei.

Atacurile bazate pe *flooding* sunt în general combinate cu un atac de tip *spoofing*, prin care se generează adrese sursă fictive şi diferite pentru pachetele din *flood*. În lipsa unui atac *spoofing*, o filtrare pe baza depăşirii unui număr limită de pachete per sursă ar elimina atacul *flooding*. Atacul *spoofing* asigură în acelaşi timp ascunderea identităţii atacatorului. În plus, dacă destinaţia atacului bazat pe *flooding* nu este o sursă ţintă ci adresa de difuzare a unei reţele, toate echipamentele din respectiva reţea vor încerca să răspundă unei surse ce nu există în realitate, atacul devenind un atac de tip DDoS şi se va numi un atac *smurf*.

Atacurile DDoS pornesc de la puterea enormă de calcul disponibilă în reţelele actuale locale. Unele dintre aceste atacuri pot fi neintenţionate, precum în cazul *Slashdot effect*. Numele vine de la vestitul site Slashdot, care a postat un *link* către un site cu capabilităţi mai mici. Când foarte mulţi utilizatori au încercat sa acceseze respectivul site, efectul a fost echivalent cu un atac *SYN flood*.

Atacurile DDoS se bazează în general pe infectarea unui număr cât mai mare de staţii, şi coordonarea unui atac către aceeaşi ţintă. Un astfel de atac a fost generat de virusul MyDoom.

Tot în categoria DDoS intră şi viruşi de tip IRCBots, programe care după infectarea unei staţii vor iniţia o conexiune către un server de IRC pe un canal privat, atacatorul putând să controleze toate staţiile infectate. Un IRCBot va deschide şi portul 6667 (portul implicit pentru IRC) pe maşina infectată.

Atacurile de obţinere a accesului au impactul cel mai mare, deoarece prin obţinerea accesului la un serviciu cel mai adesea pot fi compromise şi alte servicii sau maşini.

O pondere importantă în atacurile de obţinere a accesului o au atacurile bazate pe exploatarea vulnerabilităţii aplicaţiilor web. Subiectul este unul amplu, iar cei ce vor să urmărească numeroasele metode de exploatare a riscurilor de securitate ar trebui să înceapă cu cartea lui Joel Scambray: *Hacking Exposed Web Applications, Second Edition*.

Atacurile de obţinere a accesului pot avea drept ţintă şi tehnologiile din spatele aplicaţiilor web, un număr important de atacuri fiind direcţionat împotriva bazelor de date. Cel mai cunoscut astfel de atac este Inserarea de cod SQL (*SQL Injection*). Acest atac se bazează pe modul direct de interogare a bazei de date, astfel dacă interogarea se face:

```
SELECT X from TABLE where
    user = $user_input AND
    pass = $pass_input
```

La introducerea în câmpul utilizator a unui nume valid de utilizator urmat `"OR --"` (în SQL -- defineşte un comentariu), operaţia de selecţie devine:

```
SELECT X from TABLE where
    user = $xxx OR -- AND
    pass = 'nu_conteaza'
```

şi va fi validată pertru orice nume de utilizator aflat în baza de date. Mergând mai departe se poate introduce în câmpul utilizator: `"xxx OR 1=1 OR --"`, aceasta garantând accesul indiferent dacă utilizatorul xxx există sau nu în baza de date.

Pentru a preveni un astfel de atac interogarea bazei de date trebuie făcută prin funcţii de bibliotecă şi nu direct prin select.

Există aplicaţii ce oferă posibilitatea lansării unui număr mare de atacuri. O astfel de aplicaţie este metasploit (http://www.metasploit.com). Astfel, după o fază de determinare a vulnerabilităţilor (efectuată cu `nessus`) se pot selecta vulnerabilităţile găsite şi pot fi lansate simultan mai multe atacuri de obţinere a accesului.

Nu putem încheia acest subcapitol fără a atrage din nou atenţia asupra riscului de securitate adus de utilizatorii neglijenţi din reţea. Chiar şi în cazul unor utilizatori responsabili, nu trebuie ignorate riscurile unui atac bazat pe inginerie socială. Nu este o întâmplare că cel mai cunoscut

hacker al tuturor timpurilor este Kevin Mitnik, un om cu cunoştinte tehnice limitate, dar cu o bună înțelegere a modului de gândire a utilizatorilor şi a administratorilor din rețelele actuale.

9.4.2 Viruşi, viermi, troieni

Un virus este un program sau doar o secvență de cod ce se ataşează altor fişiere executabile fără cunoştința sau acceptul utilizatorului. Un virus include alături de mecanismele de execuție şi modalități de replicare inserate în codul altei aplicații.

Clasificarea viruşilor în funcție de modul de replicare distinge între mai multe tipuri de viruşi, dintre care cei mai întâlniți sunt: viruşi de email, de macro, bombe logice, viruşi de boot, viermi, şi troieni.

Un lucru important de precizat este că simpla prezență a unui fişier infectat pe un calculator nu este echivalentă cu infectarea sistemului. Pentru a infecta sistemul un virus trebuie să fie executat cel puțin odată. Din păcate unele aplicații rulează cod executabil în mod automat (fără informarea utilizatorului)

Email-ul este unul dintre modurile principale de propagare a viruşilor. **Viruşii de email** sunt conținuți în ataşamentul emailului, deseori având extensia schimbată (cel mai adesea în extensie specifică imaginilor). Aplicația țintă a unui astfel de virus este clientul de email. Datorită numărului mare de utilizatori Outlook o mare parte a acestor viruşi sunt dezvoltați special pentru această aplicație.

Odată executat un ataşament de email infectat, virusul se încarcă în memoria RAM şi va urmări să se multiplice. Pentru aceasta va căuta lista de adrese construite de clientul de email (*address book*). Majoritatea aplicațiilor client de email creează lista de adrese în mod dinamic, fără consultarea utilizatorului, şi nu o protejează prin criptare. Odată deschisă lista de adrese virusul va folosi direct API-ul de trimis emailuri pentru a se multiplica. Exemplele de viruşi de email sunt multiple: *MyDoom*, *I love you*, etc

O a doua categorie importantă de viruşi o reprezintă **viruşii de macro**. Un macro este o bucată de cod executabil, ataşată unui fişier document. Aplicațiile țintă a acestor viruşi sunt cele de gestionare a documentelor, gen editoare de text, foi de lucru, sau editoare de prezentări. Aceste aplicații în general vor atenționa utilizatorul înainte de deschiderea unui document ce conține macro-uri.

Pentru a reduce gradul de risc, dacă la deschiderea unui fişier sunteți avertizați de existența unor macro-uri, închideți documentul şi rulați un program antivirus pe respectivul fişier.

Viruşii de boot nu reprezintă o amenințare majoră pentru sistemele actuale datorită dificultății de lansare în execuție. Pentru a lansa un astfel de virus, sistemul trebuie să încerce inițializarea de pe o partiție infectată. Impactul acestor viruşi s-a diminuat odată cu dispariția *floppy* discurilor. În prezent viruşii de boot se pot propaga ori prin CD-uri, ori prin flash carduri.

Troienii (*trojan horses*) reprezintă aplicații (aparent legitime) ce conțin (sau instalează) un program maliții os. Pentru a evita astfel de atacuri se recomandă instalarea aplicațiilor doar din locații sigure, precum şi verificarea integrității fişierelor înainte de instalare.

Viermii (*worms*) sunt programe ce folosesc vulnerabilități în diferite servicii din Internet pentru a se multiplica. Viruşii de mail reprezintă o categorie de viermi.

Pentru a reduce impactul viruşilor sunt importante atât măsurile luate la nivelul rețelei, cât şi exersarea unor deprinderi din partea utilizatorilor.

Astfel, este important ca accesul în rețea să fie protejat de un firewall (dispozitiv de filtrare atât a traficului de intrare în rețea, cât şi celui de ieşire), ca serverul de email să aibă instalat un antivirus, să existe o politică de monitorizare a serviciilor, etc.

La nivelul utilizatorului este important să nu lanseze în execuție fişiere primite prin email, să nu deschidă ataşamente primite de la necunoscuți, să mențină un antivirus actualizat pe stația de lucru, etc.

9.4.3 Prevenirea atacurilor

Securizarea unui sistem sau a unei reţele pot presupune investiţii importante atât în echipamente, cât şi în software. Cu toate acestea, orice soluţie de securitate nu trebuie să piardă din vedere câteva componente de bază.

Primul pas în orice soluţie de securitate o reprezintă **stabilirea unor politici clare de securitate**. O astfel de politică de securitate va urmări:

- separarea ariilor cu nivel de securitate diferit
- definirea clară a drepturilor fiecărui utilizator
- definirea serviciilor ce trebuie oferite de fiecare componentă a reţelei

Odată stabilită politica de securitate va trebui să restricţionăm traficul ce nu este prevăzut de aceasta prin **configurarea politicilor de filtrare a pachetelor.**

Infectarea staţiilor dintr-o reţea locală poate oferi o poartă de acces unui atacator în spatele firewall-ului, reducând gradul de securitate al reţelei. Din acest motiv un administrator responsabil nu se va ocupa doar de configurarea firewall-ului şi a serverelor, dar şi de securizarea staţiilor de acces. Cel mai important pas în acest sens o reprezintă **configurarea** şi întreţinerea **programelor antivirus**.

O componentă importantă a securităţii o reprezintă confidenţialitatea. În scopul protejării unor date sensibile se recomandă **configurarea criptării traficului**. Nu trebuie pierdut din vedere consumul mare de resurse (mai ales procesor) presupus de operaţia de criptare.

9.5 Monitorizarea de reţea

Autorii de cărţi de reţele de calculatoare sau de ghiduri pentru administratorii de reţea precizează în general că activitatea de monitorizare este una dintre cele mai importante sarcini ale unui administrator. Cu toate acestea adesea, recomandările propuse sunt deseori limitate la cele evidente precum: "Un administrator trebuie să citească regulat jurnalele de pe fiecare server" sau "Este important să nu uitaţi să porniţi serviciul de monitorizare".

O explicaţie a tratării sumare a problematicilor de monitorizare se regăseşte în natura dual㠹 a subiectului: monitorizarea este nu doar un instrument de administrare a reţelei, ci şi una din armele de bază ale unui hacker. Din păcate competiţia dintre administratorul de reţea şi un atacator nu poate fi câştigată prin omiterea subiectului din cărţile de reţele de calculatoare, o astfel de omisiune aducând doar un dezavantaj administratorilor de reţea cu mai puţină experienţă.

Monitorizarea nu se rezumă doar la rularea unui demon de genul syslogd sau la deschiderea săptămânală a unei aplicaţii precum "Event Viewer". Monitorizarea începe din etapa de proiectare şi necesită alegerea unei tactici la fiecare dintre nivelurile OSI.

Cele două funcţii principale ale monitorizării sunt: asigurarea dinamică a securităţii prin alertarea în faţa unei încercări de atac, şi posibilitatea localizării şi definirii unei defecţiuni sau a unei funcţionări suboptimale.

Există mai multe criterii pentru a clasifica diversele tipuri de acţiuni de monitorizare. O primă clasificare face distincţia între monitorizarea unui dispozitiv specific, monitorizarea conectivităţii şi monitorizarea traficului din reţea.

Monitorizarea unui dispozitiv specific presupune rularea unui server ce va urmări apelarea celorlalte servicii, precum şi accesul la jurnalele create de acest server.

Monitorizarea conectivităţii se bazează deseori pe mecanisme foarte simple, cel mai important dintre acestea fiind folosirea utilitarului ping, metoda fiind denumită *ping sweep*. Crearea unui sistem de interogări ICMP periodice poate oferi administratorului informaţii despre întreruperea unui segment de reţea sau despre defectarea sau stingerea unuia dintre

servere. Un astfel de sistem de interogări poate fi realizat ori printr-un script simplu, ori se poate apela la aplicații dedicate.

Monitorizarea de rețea cuprinde numeroase componente: de la dispozitive hardware dedicate (denumite generic IDS – *Intrusion Detection System*), la programe de interceptare a traficului în rețea (programe de *sniffing*), precum și programe de evaluare a securității unui sistem din rețea, în această categorie intrând programele de scanat porturi.

Capitolul de față își propune tratarea aplicațiilor de scanat porturi, existând în secțiunea "Studii de caz" și o prezentare a celei mai populare soluții de sniffing: *Wireshark*. Tratarea soluțiilor de IDS depășește granițele acestei cărți.

9.5.1 Interceptarea traficului din rețeaua locală

Apariția switchurilor în rețelele locale a transformat mediul partajat al Ethernetului într-un mediu dedicat. Cu alte cuvinte, într-o rețea cu switchuri, pachetele schimbate între două stații nu vor mai fi vizibile de o a treia stație aflată în aceeași rețea locală.

Astfel un administrator de rețea ce dorește monitorizarea traficului către o anumită destinație va trebui ca înainte de a lansa programul de interceptare a traficului să facă un atac *ARP poisoning* pentru respectiva destinație. Pentru aceasta poate folosi orice generator de pachete, cele mai utilizate fiind: Cain, dsniff, IPSorcery.

9.5.1.1 tcpdump

`tcpdump` este un utilitar destinat monitorizării traficului în rețea și achiziționării de date. A fost inițial dezvoltat de un grup de cercetare în cadrul Lawrence Berkeley National Laboratory al Universității din California în iunie 1992.

Plăcile de rețea, cum sunt cele Ethernet, capturează la nivelul legătură de date doar cadrele adresate lor sau pe cele de difuzare. De aceea, pentru a captura toate cadrele, tcpdump trebuie să treacă interfața într-un mod special de lucru, numit *promiscuous mode*. Pentru aceasta are însă nevoie de suportul sistemului de operare.

Cel mai simplu mod de a începe monitorizarea este invocând comanda:

```
[root@frodo:~]# tcpdump
tcpdump: listening on eth0
14:45:09.252803 frodo.noi.39993 > 64.12.30.90.5190: P 444608787:444608793(6) ack 1453900500 win
32893 (DF)
14:45:09.254097 frodo.noi.33335 > main.noi.domain: 16713+ PTR? 90.30.12.64.in-addr.arpa.(42)
(DF)
14:45:09.254583 main.noi.domain > frodo.noi.33335:  16713 NXDomain 0/1/0 (113)(DF)
```

Să urmărim o analiză sumară a output-ului de mai sus:

- 14:45:09.252803 - timpul în care pachetul TCP a intrat în rețeaua locală (nu face parte din pachet)
- frodo.noi.39993 - sursa și portul sursă
- 64.12.30.90.5190 - destinația și portul destinație
- P, ack 1453900500, DF - proprietăți TCP
- 444608787:444608793(6) - intervalul numerelor de secvență, urmat de numărul de octeți din interval
- win 32893 - dimensiunea ferestrei pe care sursa e capabilă să o accepte

Pentru a vedea și conținutul pachetului utilizăm opțiunea -x:

```
root@frodo:~# tcpdump -c1 -x
tcpdump: listening on eth0
18:38:51.087244 64.12.24.48.5190 > frodo.noi.32777: P 3127656695:3127656809(114) ack 3009470569
win 16384 (DF) [tos 0x80]
                         4580 009a c7a1 4000 6406 354d 400c 1830
                         c0a8 000b 1446 8009 ba6c 40f7 b360 e069
                         5018 4000 1f0d 0000 2a02 20dc 006c 0003
                         000b 0000 87d9 0d61 0836 3032 3033 3631
                         3400 0000 0700 0100 0200 7000 0c00 2500
```

```
                        0000
6 packets received by filter
0 packets dropped by kernel
```

Conform structurii unui pachet IP, primii 20 octeți (4580 009a c7a1 4000 6406 354d 400c 1830 c0a8 000b) reprezintă antetul IP iar următorii 20 de octeți (1446 8009 ba6c 40f7 b360 e069 5018 4000 1f0d 0000) reprezintă antetul TCP.

Putem verifica câteva dintre aceste numere. A doua jumătate a celui de-al 5-lea grup de doi octeți (6406), 06, reprezintă protocolul încapsulat în pachetul IP, în cazul nostrul TCP (adică 6). Numerele 9 şi 10 ar trebui să codifice adresa destinație. Dacă transformăm c0a8 000b în baza 10 obținem 192.168.0.11, adică exact adresa lui frodo.noi.

În ceea ce priveşte antetul TCP, primii 2 octeți (1446), transformați în baza 10 reprezintă exact portul sursă: 5190 = 1 * 4096 + 4 * 256 + 4* 16 + 6.

Există soluții grafice ce realizează interpretarea conținutului pachetului. Cea mai populară aplicație grafică de interceptare şi interpretare a traficului de rețea este *Wireshark*, utilitar disponibil atât în mediul Windows, cât şi în Linux.

`tcpdump` oferă un grad ridicat de flexibilitate. Administratorii de rețea experimentați pot automatiza operații folosind scripturi bazate pe `tcpdump`.

Se pot utiliza combinații ale parametrilor împreună cu operatorii de negație (! sau `not`), şi (&& sau `and`), sau (|| sau `or`). De exemplu, pentru a se captura într-un fişier tot traficul mai puțin cel de SSH între maşinile frodo şi athos comanda va fi:

```
# tcpdump -w tracefile not "(port 22 and host frodo and host athos)"
```

9.5.2 Scanarea de porturi

*Un **program de scanat porturi** este o aplicație care inițiază conexiuni către toate porturile unei sistem țintă. Scopul este cel de a detremina care porturi sunt deschise, aflând astfel ce servicii sunt disponibile.*

Discuția despre rolul programelor de scanat porturi împarte administratorii de rețea în două tabere înverşunate: primul grup susține că numai hackerii folosesc programe de scanat porturi pentru a lansa apoi atacuri, în vreme ce a doua tabără susține că scanarea periodică a porturilor este singura garanție a securității unei maşini.

Deşi ambele tabere recunosc că principala utilitate a scanerelor de port constă în funcția de monitorizare a porturilor deschise, primul grup va găsi o lungă listă de programe alternative care rulate local pe o maşină vor întoarce lista porturilor deschise. Astfel, dacă pe un sistem a fost pornit un nou server în urma unui atac, aceste programe îl vor raporta. Totuşi, scenariul în care un atacator se mărgineşte doar să-şi deschide o portiță de intrare prin rularea unui server este mai degrabă naiv. Prima preocupare a unui hacker care a obținut accesul pe un server este aceea de a-şi camufla prezența şi acțiunile viitoare. Astfel, printre primele aplicații ce vor fi înlocuite se află executabile ca: netstat, ps, top, sau chiar tcpdump. În unele cazuri este înlocuită chiar şi imaginea de nucleu, făcând astfel imposibilă detectarea locală a startării altor servicii decât cele inițiale. Scanarea e primul pas în lansarea unui atac de obținere a accesului.

Sfârşitul anilor '90 în România a dus la o creştere dramatică a popularității programelor de scanat porturile, atât în rândul hackerilor cât şi al administratorilor de rețea. Principala cauză a acestei schimbări a fost dezvoltarea rețelei metropolitane, precum şi a diferenței foarte mari între costul traficului metropolitan şi cel al traficului Internet. Astfel într-un interval relativ scurt numărul atacurilor asupra serverelor aflate în rețele publice (mai ales asupra celor aflate în universități) a crescut spectaculos.

Deşi nu este ilegală, folosirea unui scaner de porturi poate fi interpretată drept un atac, şi prin urmare este posibil să vă fie filtrat accesul din rețeaua din care au fost inițiate scanările. O astfel de restricționare este totuşi destul de improbabilă.

9.5.2.1 Utilitarul nmap

Există numeroase aplicații de scanat porturile disponibile *shareware* sau *freeware*, una dintre cele mai populare fiind `nmap`. Datorită popularității de care se bucură utilitarul `nmap` în rândul administratorilor de sisteme Linux (www.insecure.org/nmap), aplicația a fost portată şi pe platforme Windows (sourceforge.net/projects/nmapwin).

`nmap` implementează mai multe metode de scanare, putând oferi informații despre porturile deschise, dar şi despre porturile explicit filtrate. În plus față de informațiile referitoare la sericiile rulate, `nmap` pune la dispoziția administratorului de rețea metode de aflare a versiunii sistemului de operare, versiunile serviciilor active, utilizatorii ce rulează aceste servicii, precum şi multe alte informații relevante pentru sistemul scanat.

Dintre cele opt tipuri importante de scanare, şase se referă la scanarea porturilor TCP, unul la scanarea porturilor UDP şi unul la monitorizarea conectivității, această ultimă metodă încercând să determine doar dacă stația destinație este activă şi dacă se poate ajunge la ea. O analiză detaliată a metodelor de scanare presupune o prezentare a câmpurilor de control din antetul TCP, prezentare ce depăşeşte scopul acestui capitol.

Dintre cele şase metode de scanare pentru porturi TCP singura disponibilă utilizatorilor neprivilegiați este TCP connect. TCP connect încearcă deschiderea unei conexiuni către portul țintă; dacă apelul sistem connect reuşeşte înseamnă că portul este deschis, altminteri portul este nefolosit.

TCP SYN este modul implicit de scanare pentru utilizatorii privilegiați. Se trimite un pachet SYN către maşina care este scanată. În cazul în care se primeşte drept răspuns un pachet SYN|ACK înseamnă că există un server ce ascultă pe acest port, dar dacă pachetul de răspuns este un pachet RST, înseamnă că portul în cauză nu este deschis. Dacă nu se primeşte nici un răspuns portul este considerat ca fiind filtrat de un *firewall*.

TCP SYN se bazează pe trimiterea de pachete SYN, ceea ce presupune drepturi de administrator, şi din acest motiv nu este accesibil utilizatorilor neprivilegiați.

Alte două opțiuni de scanare importante sunt `-sV` şi `-O`. Prima dintre aceste opțiuni oferă informații despre versiunea serviciilor ce ascultă pe porturile deschise, în vreme ce opțiunea `-O` oferă informații despre sistemul de operare instalat, precum şi timpul de la ultima rebootare.

```
kiwi:~# nmap -sV -O 141.85.37.53

Starting nmap 3.55 ( http://www.insecure.org/nmap/ ) at 2004-08-27 17:23 EEST
Interesting ports on dhcp-53.cs.pub.ro (141.85.37.53):
(The 1659 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 3.8.1p1 (protocol 1.99)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.5.25 - 2.6.3 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)
Uptime 0.105 days (since Fri Aug 27 14:52:12 2004)

Nmap run completed -- 1 IP address (1 host up) scanned in 11.840 seconds
```

Un lucru ce nu trebuie trecut cu vederea este faptul că procesul de scanare de porturi poate duce la o încărcare semnificativă a sistemului, o operație de monitorizare putând avea efectele unui atac DoS.

9.5.2.2 Funcționarea nmap

Scanarea ACK are drept principal scop determinarea tipurilor regulilor de filtrare folosite de un firewall ce protejează destinația. La finalul unei astfel de scanări poate fi aflat dacă un port destinație este filtrat de o regulă ce foloseşte monitorizarea stărilor conexiunilor sau nu.

În acest scop scanarea ACK trebuie rulată în conjuncție cu scanarea TCP SYN. Mai exact, pentru un port ce a fost determinat ca port filtrat în urma unei scanări TCP SYN se va rula scanarea ACK pentru a determina tipul regulii de filtrare folosită.

Pentru a verifica funcționarea scanării vom urmări pachetele trimise în rețea pentru patru cazuri: port închis, port deschis, filtrarea doar a pachetelor de SYN, filtrarea cu o regulă ce monitorizează starea conexiunii.

Cazul 1. Portul 10000 este închis. Procesul de scanare a traficului prin:

```
kiwi:~# tcpdump -i eth1 tcp port 10001
```

Scanarea portului 10001 prin metoda TCP SYN se va face cu comanda:

```
cursuri:~# nmap -sS -p 10001 -P0 -n 141.85.37.145

Starting nmap V. 2.54BETA31 ( www.insecure.org/nmap/ )
The 1 scanned port on  (141.85.37.145) is: closed

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

Pachetele trimise în rețea au fost prinse de tcpdump:

```
  17:11:42.860449 IP cursuri.cs.pub.ro.54403 > kiwi.cs.pub.ro.10001: S
2570971707:2570971707(0) win 2048
  17:11:42.860549 IP kiwi.cs.pub.ro.10001 > cursuri.cs.pub.ro.54403: R
0:0(0) ack 2570971708 win 0
```

Rezultatul scanării cu ACK-uri va fi următorul:

```
cursuri:~# nmap -sA -p 10001 -P0 -n 141.85.37.145

Starting nmap V. 2.54BETA31 ( www.insecure.org/nmap/ )
The 1 scanned port on  (141.85.37.145) is: UNfiltered

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

Scanarea de pachete a interceptat 2 pachete pentru portul în cauză: segmentul ACK trimis de nmap şi segmentul RST trimis drept răspuns.

```
  17:18:04.102979 IP cursuri.cs.pub.ro.38388 > kiwi.cs.pub.ro.10001: . ack
1214695285 win 4096
  17:18:04.103089 IP kiwi.cs.pub.ro.10001 > cursuri.cs.pub.ro.38388: R
1214695285:1214695285(0) win 0
```

Cazul 2. Portul deschis va fi simulat prin comanda nc:

```
kiwi:~# nc -l -p 10000
```

Rezultatul scanării ACK va fi acelaşi ca şi în cazul portului închis, diferențe apărând doar în scanarea TCP SYN:

```
cursuri:~# nmap -sS -p 10001 -P0 -n 141.85.37.145

Starting nmap V. 2.54BETA31 ( www.insecure.org/nmap/ )
Interesting ports on  (141.85.37.145):
Port       State       Service
10001/tcp  open        unknown

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

Rezultatul tcpdump va fi:

```
  17:19:29.123875 IP cursuri.cs.pub.ro.63326 > kiwi.cs.pub.ro.10001: S
680066281:680066281(0) win 3072
  17:19:29.124006 IP kiwi.cs.pub.ro.10001 > cursuri.cs.pub.ro.63326: S
1412873701:1412873701(0) ack 680066282 win 5840 <mss 1460>
  17:19:29.124164 IP cursuri.cs.pub.ro.63326 > kiwi.cs.pub.ro.10001: R
680066282:680066282(0) win 0
```

9.6 Studiu de caz

9.6.1 Drepturile pe fişiere in NTFS

Sistemul de fişiere NTFS oferă o serie de îmbunătăţiri atât din punctul de vedere al performanţei, cât şi al securităţii, faţă de predecesorii săi, variantele de sisteme de fişiere FAT, în care toate conturile de utilizator au drepturi egale şi depline asupra sistemului de fişiere.

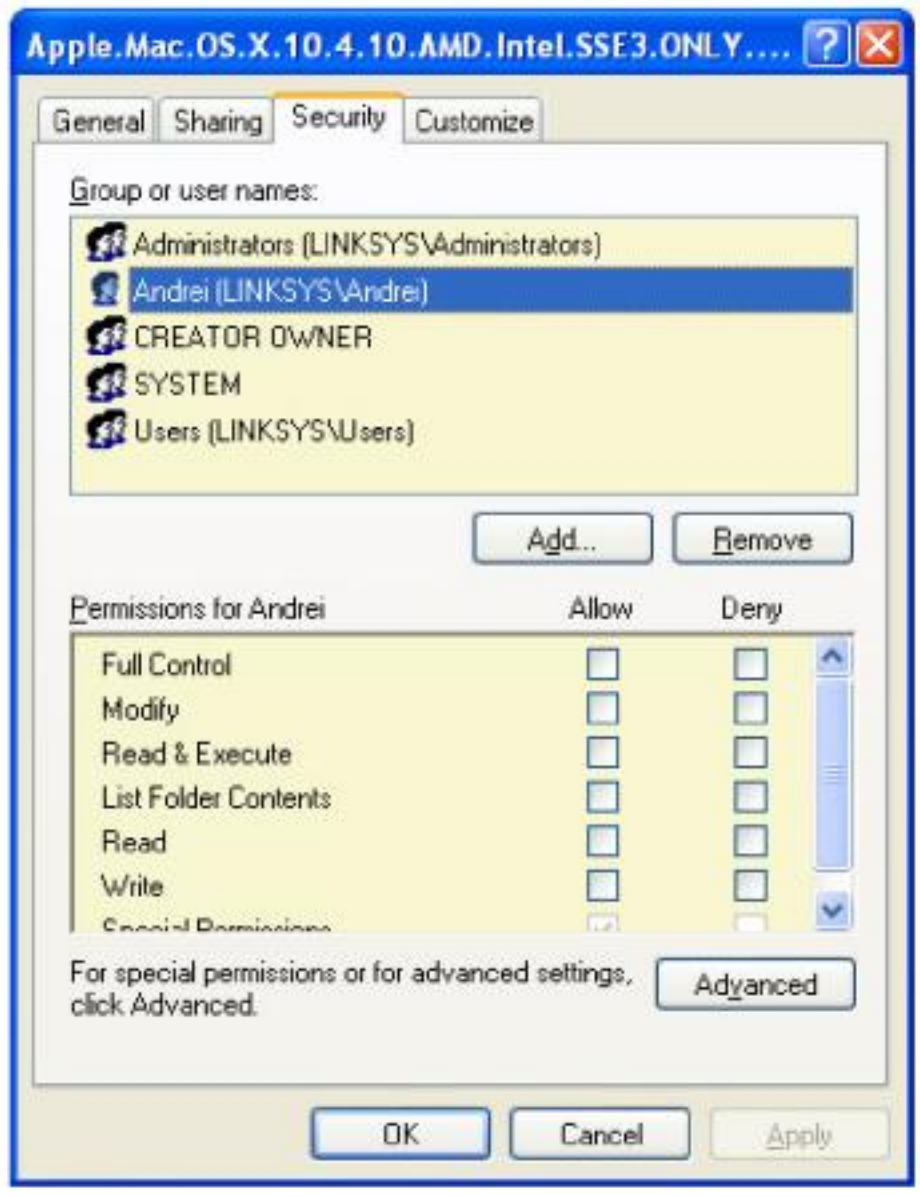


Figura 9-1 Precizarea drepturilor în NTFS

Securitatea în NTFS se implementează prin intermediul unor liste de acces care sunt, practic, nişte descriptori ataşati fişierelor şi directoarelor din sistemul de fişiere. Pentru ca un proces sa poată accesa un fişier sau un director, sistemul de securitate verifică dacă procesul are dreptul de a-l accesa. O variantă centralizată pentru implementarea securităţii pe NTFS este folosirea lui Active Directory (în Windows) prin care staţiile sunt grupate pe domenii şi folosesc o autentificare bazată pe un server extern pentru a implementa şi a controla securitatea pe staţiile individuale.

Ca măsură sporită de securitate, sistemul de fişiere NTFS oferă posibilitatea criptării datelor pe care le stochează. Avantajul major al implementării mecanismelor de criptare direct în sistemul de fişiere îl reprezintă viabilitatea pe care acest tip de securitate o menţine în lucrul cu orice sistem de operare care suporta sistemul respectiv de fişiere.

Din mediul Windows, criptarea unui fişier sau a unui director (inclusiv a conţinutului său) se face din ecranul de proprietăţi al fişierului/directorului (clic dreapta > Properties) > Advanced > „Encrypt contents to secure data".

În Windows (XP, pentru exemplul de faţă) securitatea fişierelor poate fi implementată şi pentru fiecare utilizator în parte. Cu alte cuvinte, pe lângă drepturile de acces şi interacţiune cu sistemul de operare pe care un utilizator le poate avea sau îi pot fi suprimate, acestora li se poate oferi acces (eventual limitat) asupra fişierelor şi directoarelor din sistemul de fişiere.

Pentru a putea acţiona granular asupra securităţii sistemului de fişiere, în Windows trebuie dezactivată opţiunea „Use simple file sharing" (de la Explorer > Folder Options > View > Use Simple File Sharing: debifare). Ulterior, atribuirea drepturilor se face din fereastra de proprietăţi a unui director sau a unui fişier, sub tabul „*Security*". Se selectează utilizatorul sau grupul care va fi editat, iar din lista de drepturi se selectează indică permisiunile ce vor fi atribuite. Atenţie, toate configurările se aplică doar fişierului sau directorului selectat. Pentru a configura mai explicit permisiunile, butonul „*Advanced*" afişează o noua interfaţă în care pot fi vizualizate toate tipurile de permisiuni ce sunt aplicate fişierului sau directorului ales, inclusiv drepturi moştenite prin ierarhia sistemului de fişiere

9.6.2 Recuperarea parolei

Un bun administrator de reţea trebuie să fie conştient de mecanismele de recuperare a parolei pentru a putea preveni proceduri neautorizate de recuperare a parolei de root.

Există două metode importante pentru a realiza recuperarea parolei de root, ambele presupunând accesul fizic la echipament. Prima metodă presupune modificarea din meniul bootloader-ului a parametrilor trimişi la încărcarea nucleului, în vreme ce a doua se bazează pe folosirea unui CD Linux bootabil.

9.6.2.1 Editarea grub

Astfel, dacă atacatorul are acces fizic la server, acesta poate reboota sistemul. La bootarea sistemului va fi iniţializat bootloader-ul. Din meniul de grub se selectează imaginea de nucleu ce se vrea încărcată şi se activează editarea respectivei linii de grub prin opţiunea `"e"`. În capitolul 5 sunt prezentate pe larg metodele de modificare a opţiunilor transmise la încărcarea nucleului.

La imaginea de nucleu selectată se adaugă opţiunea `init=/bin/bash` şi se continuă procesul de iniţializare a sistemului. Ca variante pentru acest pas putem folosi şi opţiunile: `"init S"` sau `"init=1"`.

În urma încheierii procesului de iniţializare se va obţine un shell root.

Dacă la execuţia comenzii `passwd` se primeşte mesaj de eroare trebuie remontat sistemul de fişiere pentru scriere şi citire. Acest lucru se realizează folosind comanda:

```
# mount -o remount,rw /
```

După remontare vom putea executa comanda `passwd`.

Pentru a preveni o recuperare neautorizată a parolei de root se recomandă folosirea parolelor pentru bootloader.

9.6.2.2 Folosirea unui CD bootabil

A doua metodă de recuperare a parolei presupune folosirea unui CD Linux bootabil. Dacă ordinea de boot nu acordă prioritate unității optice în fața hard discului, va trebui schimbată ordinea de boot din BIOS.

După inițializarea sistemului de pe CD trebuie obținut un shell de root. Pentru ubuntu aceasta se realizează prin executarea din contul neprivilegiat a comenzi `sudo`:

```
# sudo bash
```

Trebuie creat un nou director în care se va face montarea vechiului sistem de fişiere:

```
# mkdir /mnt/x
# mount /dev/hda2 /mnt/x
```

Putem verifica că sistemul montat are baza de date de utilizatori a sistemului inițial:

```
# cat /mnt/x/etc/passwd
# ls -l /mnt/x/etc/passwd
```

În acest moment putem invoca comanda `chroot`:

```
# chroot /mnt/x
```

Baza de utilizatori are acum aceeaşi dimensiune cu cea inspectată mai devreme:

```
# ls -l /etc/passwd
```

Ultimul pas presupune invocarea utilitarului `passwd` şi schimbarea parolei de root.

Pentru a preveni o astfel de procedură este necesar să forțăm din BIOS ca prim dispozitiv de boot unul dintre hard discuri, iar apoi să prevenim modificarea opțiunilor de BIOS prin definirea de parole de BIOS.

9.6.3 Limitarea spațiului pe disc pentru utilizatori: quota

Folosind programul `quota`, putem stabili un set de limite pentru spațiul alocat fiecărui utilizator pe disc. Un administrator poate proteja astfel sistemul de un utilizator care ocupă prea mult spațiu fizic pe disc, sau poate folosi utilitarul pentru monitorizarea sistemului şi a spațiului ocupat. Putem determina acest lucru şi cu comanda `df`, însă vom obține doar o imagine de moment (*snapshot*) a discului nostru şi nu o evoluție în timp a alocării blocurilor de date. Deşi pentru un utilizator normal, nu există nevoia folosirii `quota`, utilitarul îşi regăseşte mult mai bine aplicabilitatea în întreținerea serviciilor de *web hosting* sau în aplicațiile de *project traking*. Există două tipuri de limite (denumite prin abuz "quota") ce se pot impune pe un sistem de fişiere:

- block quota – limitează spațiul ce poate fi ocupat de un utilizator/grup de utilizatori;
- file quota – limitează numărul de fişiere ce se pot crea de către un utilizator/grup de utilizatori (aici termenul de fişier acoperă orice fel de entitate: director, fişier special, link-uri spre alte fişiere);

Înțelegerea termenului de limită software este foarte importantă. Quota implementează această limită ca un prag virtual de dimensiune mai mică decât dimensiunea totală a spațiului fizic alocat. Odată ce utilizatorul a depăşit acest prag el intră într-o perioadă de grație ("grace period"), inițial de 7 zile, în care trebuie să îşi şteargă din spațiul ocupat pentru a reveni sub limită. Dacă nu face acest lucru, limitarea software se transformă într-o limitare hardware şi utilizatorul nu va mai putea crea fişiere noi pe respectivul sistem de fişiere. De asemenea, concomitent cu depăşirea limitei software, administratorul este anunțat de acest fapt. O `quota` se adaugă pentru fiecare sistem de fişiere, grup, şi utilizator separat.

Instalare

```
waters@kiwi:~$ sudo apt-get install quota
```

Dacă utilitarul `apt-get` nu găseşte pachetul `quota`, vor trebui adăugate în `/etc/apt/sources.list` liniile următoare :

```
deb http://ro.archive.ubuntu.com/ubuntu/ feisty universe
deb-src http://ro.archive.ubuntu.com/ubuntu/ feisty universe
deb http://ro.archive.ubuntu.com/ubuntu/ feisty multiverse
deb-src http://ro.archive.ubuntu.com/ubuntu/ feisty multiverse
```

Configurare

Se editează fişierul `/etc/fstab` pentru a putea aplica limitele pe fiecare sistem de fişiere.

```
waters@kiwi:/etc$ sudo vim fstab
```

Se adaugă opţiunile de `usrquota` şi `grpquota` pentru sistemul de fişiere pe care se vrea să se folosească quota. Chiar dacă nu se va folosi utilitarul pentru a impune restricţii pe grupuri de utilizatori, este recomandată adăugarea opţiunii de `grpquota` pentru a evita posibilele avertismente. Spre exemplu, pentru a aplica limitări pe partiţia /dev/sda6 trebuie adăugat în fişierul `/etc/fstab`:

```
# /dev/sda6
UUID=6c05a9f2-1eac-4d7b-9b75-0c5bc1067dcf / ext3 defaults, errors=remount-
ro,usrquota,grpquota 0 1
```

Pentru a încărca noile opţiuni, partiţia trebuie remontată:

```
waters@kiwi:/etc$ sudo mount -o remount /
```

Aplicarea `quota` de către sistemul de operare pe sistemul de fişiere dorit se poate verifica prin comanda:

```
waters@kiwi:/etc$ mount | grep quota
/dev/sda6 on / type ext3 (rw,errors=remount-ro,usrquota,grpquota)
```

Pornirea utilitarului pe sistemul de fişiere se face prin comanda:

```
waters@kiwi:/$ sudo /etc/init.d/quota start
* Checking quotas... [ OK ]
* Turning on quotas... [ OK ]
```

Rularea scriptului quota realizează verificarea şi pornirea `quota`. Acelaşi efect s-ar fi obţinut rulând comenzile `sudo quotacheck -augmv` (pentru verificare) şi `sudo quotaon -augv` (pentru pornire). La pornirea utilitarului, în timpul primei verificări, `quota` va crea în punctul de montare specificat în `/etc/fstab`, două fişiere numite: `aquota.user`, `aquota.grup` pentru versiunea a 2-a de quota. sau `quota.user`, `quota.group` pentru versiunea 1.

Utilizarea quota

Comanda:

```
waters@kiwi:~$ sudo edquota -u waters
```

va deschide un editor de texte (mai precis editorul text referit de variabila de mediu $EDITOR), în care se vor putea seta limitele quota:

```
GNU nano 2.0.2 File: /tmp//EdP.aA5wZJx

Disk quotas for user waters (uid 1000):
Filesystem blocks soft hard inodes soft hard
/dev/sda6 20976 0 0 712 0 0
```

În exemplul de mai sus este prezentat sistemul de fişiere asupra căruia este aplicată quota, numărul de blocuri ocupate în prezent (aprox 20 MB), şi numărul de inodes (fişiere) existente în prezent. Limitele *soft* şi *hard* pentru ambele quote, blocks sau inodes, sunt setate pe 0; deci nu este impusă nici o restricţie. Limita soft reprezintă pragul care, odată depăşit de utilizator, va declanşa intrarea în perioada de graţie. Limita hard reprezintă pragul superior permis, care nu poate fi depăşit fizic pe disc.

Pentru de graţie iniţială este de 7 zile. Pentru a o schimba folosim comanda:

```
waters@kiwi:~$ sudo edquota -t
```

```
GNU nano 2.0.2 File: /tmp//EdP.aC0aFRq

Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
Filesystem Block grace period Inode grace period
/dev/sda6 7days 7days
```

Monitorizarea prin quota:

Se pot genera rapoarte periodice folosind comanda:

```
waters@kiwi:/etc$ sudo repquota -as
*** Report for user quotas on device /dev/sda6
Block grace time: 7days; Inode grace time: 7days
Block limits File limits
User used soft hard grace used soft hard grace
----------------------------------------------------------------------
root -- 2272M 0 0 90319 0 0
daemon -- 56 0 0 5 0 0
man -- 428 0 0 16 0 0
news -- 4 0 0 1 0 0
dhcp -- 4 0 0 4 0 0
klog -- 4 0 0 2 0 0
avahi-autoipd -- 12 0 0 3 0 0
avahi -- 4 0 0 1 0 0
cupsys -- 72 0 0 16 0 0
haldaemon -- 4 0 0 1 0 0
hplip -- 4 0 0 1 0 0
waters -- 20976 0 0 712 0 0
```

Exemplu de depăşire a limitei quota:

După ce am setat limita hard a quotei la 200M pe sistemul ext3,

```
/dev/sda6 20976 100 200 712 0 0
```

(vezi rezultatul comenzii `sudo edquota -u`)

efectuăm operaţiile:

```
waters@kiwi:/media/sda5/x$ ls -lh
total 900M
-r-xr-x--- 1 root plugdev 573M 2006-12-02 16:11 image.nrg
waters@kiwi:/media/sda5/x$ cp image.nrg /home/waters
cp: writing `/home/waters/image.nrg': Disk quota exceeded
```

Cuvinte cheie

- Atacuri
- chmod
- chown
- chroot
- df
- DoS
- drepturi pe fişiere în NTFS
- drepturi pe fişiere în UNIX
- du
- firewall
- hacker
- netstat
- nmap
- parole
- politică de securitate
- quota
- securitatea fizică
- sniffer
- sudo
- suid
- syslog
- tcpdump
- top
- ulimit
- umask
- viruşi

Întrebari

Un utilizator primeşte un email de la utilizatorul privilegiat al domeniului cu un update de securitate ce trebuie instalat pe staţia sa. Ataşamentul este criptat, iar cheia de decriptare este precizată în corpul emailului. Serverul de mail are instalat un antivirus actualizat. Aplicaţia trebuie instalată pentru că:

- ❑ Mailul a fost semnat de administratorul sistemului
- ❑ Mailul a fost primit printr-un server ce are un antivirus actualizat
- ❑ Aplicaţia a fost trimisă criptată
- ❑ Nu trebuie instalată. Este probabil să conţină un virus.

Fie următoarele comenzi:

```
umask 022
echo "Acesta nu este un test" > test
chmod go+x test
```

Care vor fi drepturile pentru utilizatorii grupului ce aparţine fişierul test?

- ❑ doar citire
- ❑ doar execuţie
- ❑ citire şi execuţie
- ❑ citire, scriere şi execuţie

Care dintre parolele de mai jos este cea mai sigură pentru utilizatorul cornel?

- ❑ cornelush
- ❑ coRnel
- ❑ cOrn31sh
- ❑ nici una. Toate aceste parole vor fi încercate într-un atac bazat pe dicţionar.

Care din riscurile de mai jos reprezintă riscuride securitate fizică pentru un server?

- ❑ Lipsa unui firewall
- ❑ Parole pentru utilizatori prea simple
- ❑ Vârfuri de tensiune în reţeaua electrică
- ❑ Absenţa licenţei pentru sistemul de operare

În urma procesului de scanare de porturi se poate determina pentru un serviciu:

- Dacă un serviciu rulează pe un port dat
- Versiunea serviciului
- Dacă serviciul este filtrat
- Toate variantele

Primirea unui email ce conține un virus pe un sistem ce nu are antivirus duce automat la infectarea sa. Un sistem infectat cu un virus nu mai poate fi ținta unui alt atac.

- adevărat, adevărat
- adevărat, fals
- fals, adevărat
- fals, fals

Pentru un serviciu dar, care dintre următoarele contatări nu poate fi făcută folosind tcpdump?

- Portul pe care rulează serviciul
- Versiunea serviciului
- Dimensiunea pachetelor folosite pentru un anumit serviciu
- Adresa IP a serverului pe care rulează

Pentru a limita numărul de procese per utilizator se poate folosi:

- ulimit
- qouta
- nmap
- chroot

Care este primul pas în securizarea unei rețele?

- definirea unei politici de securitate
- definirea regulilor de filtrare pe firewall
- instalarea de programe antivirus pe toate stațiile din rețea
- criptarea traficului

Ce se poate deduce din rezultatul comenzii netsat?

```
cursuri:~# netstat -tapn
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      1851/apache
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      1838/proftpd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1831/sshd
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      1822/master
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      1851/apache
tcp        0      0 141.85.37.5:80          212.95.252.16:4194      TIME_WAIT   -
tcp        0      0 141.85.37.5:80          212.95.252.16:1184      ESTABLISHED14160/apache
tcp        0      0 141.85.37.5:80          193.138.193.163:34533   ESTABLISHED13932/apache
tcp        0      0 141.85.37.5:22          141.85.37.2:32844       ESTABLISHED19531/1
tcp        0      0 141.85.37.5:80          207.46.98.76:3355       TIME_WAIT   -
tcp        0      0 141.85.37.5:80          212.95.252.16:4784      TIME_WAIT   -
tcp        0      0 141.85.37.5:80          151.196.5.210:50062     TIME_WAIT   -
tcp        0      0 141.85.37.5:80          193.138.193.163:34490   ESTABLISHED14022/apache
tcp        0      0 141.85.37.5:80          212.95.252.16:2975      TIME_WAIT   -
tcp        0      0 141.85.37.5:22          141.85.37.25:4898       ESTABLISHED13941/sshd:
```

- există o sesiune de web către adresa 193.138.193.163
- pe sistemul local nu rulează un server de nume (DNS)
- există două sesiuni de ssh către maşina locală
- există o conexiune de ftp stabilită

10 Compilare, linking şi execuție

Of course, all of the software I write runs on Linux; that's the beauty of standards, and of cross-platform code. I don't have to run your OS, and you don't have to run mine, and we can use the same applications anyway!
Jamie Zawinski

Ce se învață din acest capitol?

- Cum se obține dintr-un program sursă un program executabil
- Compilarea şi interpretarea unui program
- Etapele compilării
- Utilizarea GCC pentru compilarea unui program
- Ce sunt şi la ce folosesc bibliotecile de funcții
- Concepte de portabilitate
- Cum se pot automatiza acțiuni cunoscând depențele între ele; utilitarul make
- Cum interacționează aplicațiile şi bibliotecile cu sistemul de operare

10.1 Introducere

Unul dintre aspectele importante în utilizarea unui sistem de operare este dezvoltarea de noi programe. Acest lucru înseamnă scrierea de cod sursă, compilarea şi rularea programului astfel obținut. Sistemele de operare moderne pun la dispoziția dezvoltatorului/programatorului o suită de aplicații folosite pentru scrierea de noi programe. Astfel, în cazul Windows, există suita de aplicații *Visual Studio*. Aceasta permite crearea de programe în diverse limbaje de programare şi cu utilitate diversă. Dezvoltarea de aplicații se desfăşoară într-un mediu integrate (IDE - *Integrated Development Environment*). De partea cealaltă, sistemele Linux pun la dispoziția utilizatorului suita de aplicații GNU, folosită pentru compilarea, link-editarea şi depanarea programelor.

10.1.1 Editoare

Prima fază din dezvoltare este scrierea de cod specific unui limbaj de programare. **Cod** înseamnă o înşiruire de instrucțiuni, variabile şi funcții pentru acel limbaj. Fişierul în care se scrie codul este, în general, un fişier text şi poartă numele de **fişier sursă**, sau mai simplu sursă. Expresia "trebuie să mă uit prin surse" se referă la parcurgerea fişierelor sursă ale unei aplicații pentru analiza codului.

Scrierea de cod sursă se realizează prin intermediul unui editor. Editoare populare în lumea Unix sunt vi, Emacs, kate, nano, joe. Editoare populare în lumea Windows sunt Notepad++, Ultraedit, Crimson Editor. De obicei, în lumea Windows, editoarele sunt parte a unui IDE (*Integrated Development Environment*) cum este Visual Studio sau Eclipse. Un editor este folosit pentru scrierea unui fişier sursă. Drept urmare, editoarele moderne prezintă o serie de

caracteristici utile acestui scop: indentare automata, colorare a codului (*syntax highlighting*), *code folding*, *variable completion*, folosirea de registre de editare pentru operatii multiple, etc.

Lumea Unix este dominata de rivalitatea dintre editoarele **vi** şi **Emacs**. În vreme ce vi este un adept al filozifiei Unix (*keep it simple*) şi este folosit doar pentru editarea de fişiere sursă, Emacs este un editor complex cu facilităţi care includ posibilitatea de folosire ca mediu integrat de dezvoltare (*IDE*), client de e-mail, client de IRC, etc. Ambele editoare permit posibilitatea de extensie şi personalizare. Emacs a fost proiectat special pentru a fi extensibil şi dispune de un limbaj propriu (Emacs-Lisp) folosit pentru a extinde funcţionalitatea sau pentru a îl personaliza.

10.1.2 Compilare şi interpretare

Programul scris în cod sursă este compilat sau interpretat pentru a îndeplini sarcina pentru care a fost scris. Atât **compilarea** cât şi **interpretarea** unui program sunt forme de translatare a acestuia şi execuţie pe un sistem hardware. Translatarea se referă la transformarea unui program scris într-un limbaj, într-un alt limbaj. În definiţia cea mai simplă, compilarea unui program se referă la obţinerea unui fişier binar executabil. Executarea acestui fişier duce la îndeplinirea sarcinii pentru care a fost scris programul. Vom vedea [vezi 10.2.3] că există mai multe faze specifice compilării unui program.

De partea cealaltă, interpretarea unui program elimină faza obţinerii unui executabil şi execută codul asociat unui program pe măsura parcurgerii acestuia. Avantajul interpretării unui program este timpul mai scurt de dezvoltare (se elimină faza de compilare) şi depanarea mult mai facilă. Pe de altă parte, dezavantajul interpretării unui program este viteza mai scăzută de execuţie. În cazul compilării, rularea executabilului înseamnă rularea de cod specific sistemului fizic. În cazul interpretării, codul sursă este parcurs, analizat şi apoi executat, rezultând într-un timp de rulare mai mare.

10.1.2.1 Limbaje compilate şi limbaje interpretate

Utilitarul folosit pentru compilarea unui program poartă numele de **compilator**, iar cel folosit pentru interpretare poartă numele de **interpretor**. Compilatorul, respectiv interpretorul, cunosc sintaxa limbajului de programare în care a fost scris respectivul program şi sunt folosite pentru translatarea acestuia într-o formă executabilă.

Una dintre cele mai mari confuzii din lumea limbajelor de programare şi a programării în general este clasificarea în limbaje compilate şi limbaje interpretate. Astfel, se consideră că C, C++, Ada, Fortran, Java sunt limbaje compilate, în vreme ce Perl, Python, PHP, Ruby, Lisp sunt limbaje interpretate.

De fapt, orice limbaj poate fi atât interpretat cât şi compilat. Astfel, deşi limbajul C este de obicei compilat, există şi interpretoare pentru acesta (de exemplu Ch interpreter - http://www.softintegration.com/). Drept urmare, nu există limbaje interpretate sau compilate ci implementări de compilator sau interpretor pentru un limbaj. Într-adevăr, anumite limbaje sunt cunoscute unui anumit tip de implementare (C ca având implementări de compilator, Perl ca având implementări de interpretor) şi, de aceea sunt denumite "limbaje compilate" sau "limbaje interpretate". Totuşi, această denumire este forţată şi ar trebui atribuită implementării acelui limbaj.

10.1.3 De la sursa la executabil, de la executabil la proces

După cum s-a precizat, în urma compilării unui fişier sursă se obţine un **executabil** care poate fi apoi rulat pentru îndeplinirea sarcinii pentru care a fost scris. S-a menţionat în capitolul 4 faptul că un executabil este o entitate pasivă care descrie modul în care va rula programul. Rularea efectivă se realizează în urma execuţiei programului în cadrul unui proces. Procesul

este entitatea activă în cadrul căreia se vor aloca resursele programului şi se va rula codul asociat pe procesor.

Aşa cum reiese şi din figura de mai jos, obţinerea unui executabil dintr-un fişier sursă se realizează în urma compilării acestuia din urmă. La fel, procesul este obţinut din executabil în urma execuţiei. Utilizatorul va trebui să cunoască doar limbajul de nivel înalt şi să scrie codul sursă în acest limbaj. În continuare, compilatorul va obţine executabilul; executabilul este un fişier binar care indică, în cod maşină, operaţiile descrise de utilizator. În faza de execuţie, cu ajutorul nucleului sistemului de operare se obţine un proces din executabilul obţinut anterior. Acest proces va executa efectiv pe procesor operaţiile descrise în cod maşină în zona de cod a executabilului [vezi 4.3.1].

Se observă, astfel, beneficiul utilizării unui compilator. Utilizatorul nu va trebui să aibă vederea sistemului fizic din spate, ci numai noţiuni despre limbajul folosit pentru a scrie codul sursă. Compilatorul va fi responsabil cu translatarea codului sursă din cod maşină în executabil, după care se invocă procesul care execută codul. Folosirea unui compilator aduce după sine avantajul **portabilităţii codului**: utilizatorul va scrie cod sursă portabil specific limbajului de programare, lăsând pe seama compilatorului generarea de cod binar neportabil specific sistemului fizic pe care acesta va rula. Mai multe detalii despre portabilitate sunt prezentate în secţiunea 10.5.

În exemplul de mai jos este prezentat un fişier sursă C (`hw.c`), compilarea acestuia (`gcc hw.c`), generarea unui executabil (`a.out`) şi executarea acestuia în cadrul unui proces (`./a.out`).

```
$ cat hw.c
#include <stdio.h>

int main (void)
{
        printf ("Hello, World!\n");

        return 0;
}
$ file hw.c
hw.c: ASCII C program text
$ gcc hw.c
$ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for
GNU/Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux 2.4.1,
not stripped
$ ./a.out
Hello, World!
```

10.1.4 Pachete necesare

Pentru dezvoltarea de aplicaţii este nevoie, aşadar, de cunoştinţe despre un anumit limbaj de programare, de un editor folosit pentru a scrie codul sursă şi de acces la un interpretor sau un compilator pentru acel limbaj. În restul acestui capitol, limbajul de programare folosit pentru exemplificare va fi C. Compilatorul folosit va fi compilatorul implicit pe distribuţiile Linux, anume gcc.

Pachetele necesare pentru dezvoltarea de aplicaţii C sub Ubuntu sunt `gcc`, `libc6-dev`. Se recomanda şi paginile de manual pentru dezvoltare din pachetul `manpages-dev`.

```
# apt-get install gcc libc6-dev manpages-dev
```

10.2 Compilare. GCC

În mod obişnuit, compilarea se referă la obținerea unui fişier executabil dintr-un fişier cod sursă. Acest fişier este apoi executat. Compilatorul este cel care realizează această translatare. Un compilator este o aplicație complexă care poate avea diverse constrângeri. Astfel, un compilator trebuie să obțină un cod maşină al executabilului care să fie cât mai mic, care să ruleze cât mai rapid sau care să consume cât mai puțină memorie în momentul rulării. De asemenea, un compilator trebuie să implementez o cât mai bună parte din specificațiile standardelor în rigoare. Spre exemplu, în cazul C, standardele sunt stabilite de ISO. Ultimul standard este `ISO 9899:1999` denumit şi `C99`.

Pe sistemele Unix, compilarea rezultă în generarea executabilului implicit `a.out`:

```
$ ls
hw.c
$ gcc hw.c
$ ls
a.out  hw.c
$ ./a.out
Hello, World!
```

Pe sistemele Windows compilarea se realizează, în linia de comandă, folosind `cl`; executabilul obținut are acelaşi nume cu fişierul sursă însă cu extensia .exe:

```
C:\> cl hw.c
C:\> hw.exe
Hello, World!
```

GCC este compilatorul implicit pe distribuțiile Linux şi o bună parte din alte sisteme Unix. Scris de Richard Stallman în 1989 pentru proiectul GNU, era folosit inițial doar pentru a compila programe C (*GNU C Compiler*). De-a lungul timpului s-a extins, şi în ziua de azi, poate compila programe scrise în C++, Fortran, Objectiv-C, Pascal, Java, etc. Drept urmare, denumirea sa a fost schimbată în **GNU Compiler Collection**. Pachetul GCC oferă executabile (denumite şi *front-end*-uri) pentru fiecare tip de limbaj pe care îl poate compila; astfel, `gcc` este folosit pentru a compila programe scrise în C, `g++` pentru a compila program scrise în C++, `gnat` pentru a compila program scrise în Ada, etc.

10.2.1 Utilizare GCC

La o rulare implicită a GCC, acesta generează executabilul `a.out`. Fie următorul program C:

```
$ cat hw2.c
#include <stdio.h>

main ()
{
        printf ("Hello, World!\n");
}
```

Acesta va fi compilat şi apoi rulat:

```
$ gcc hw2.c
$ ./a.out
Hello, World!
```

Se poate specifica numele executabilului care se doreşte a fi executat prin folosirea opţiunii `-o` urmată de numele executabilului:

```
$ gcc hw2.c -o my_exec
$ ./my_exec
Hello, World!
```

În general, executabilele în Unix nu au extensie. Opţiunea `-o` poate fi prezentă oriunde în linia de comandă, doar că va trebui să fie urmată de numele executabilului:

```
$ gcc -o my_other_exec hw2.c
$ ./my_other_exec
Hello, World!
```

O greşeală frecventă este folosirea opţiunii `-o` urmată chiar de numele fişierului sursă:

```
$ gcc -o hw2.c my_other_exec
my_other_exec:(.rodata+0x0): multiple definition of `_fp_hw'
/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../lib/crt1.o:(.rodata+0x0):
first defined here
[...]
```

ATENŢIE: Această greşeală are consecinţe grave pentru că duce la pierderea fişierului sursă.

După cum se observă, programul de mai sus se compilează, dar nu respectă întru totul standardul C. De exemplu, funcţia `main` trebuie să întoarcă `int`. Compilatorul nu va afişa, în mod implicit, nici un fel de avertisment pentru astfel de inconsistenţe. Acest lucru poate fi schimbat prin folosirea opţiunii `-Wall` (*Warnings All*) care forţează compilatorul să afişeze mesaje de avertizare pentru abateri de la standard sau pentru formulări atipice:

```
$ gcc -Wall hw2.c -o my_exec
hw2.c:4: warning: return type defaults to 'int'
hw2.c: In function 'main':
hw2.c:6: warning: control reaches end of non-void function
```

După corectare, avertismentele nu vor mai fi furnizate:

```
$ cat hw2.c
#include <stdio.h>

int main (void)
{
        printf ("Hello, World!\n");

        return 0;
}
$ gcc -Wall hw2.c -o my_exec
```

O greşeală frecventă care poate fi eliminată prin folosirea opţiunii `-Wall` este o construcţie realizată astfel: `if (a = 1)`. În exemplul de mai jos se foloseşte o astfel de construcţie care este raportată de compilator în momentul folosirii opţiunii `-Wall`:

```
$ cat hw2.c
#include <stdio.h>

int main (void)
{
        int a = 1;

        if (a = 1)
                printf ("Hello, World!\n");

        return 0;
}
$ gcc -Wall hw2.c -o my_exec
hw2.c: In function 'main':
hw2.c:7: warning: suggest parentheses around assignment used as truth
value
```

NOTĂ: Din cauza sintaxei extrem de relaxate a limbajului C este recomandată folosirea opțiunii `-Wall` la orice compilare de program.

10.2.1.1 Compilare din surse multiple

O aplicație complexă va avea de obicei mai multe fișiere sursă conținând diverse părți din implementare. În exemplul de mai jos, fișierul `main.c` conține funcția `main`, fișierele `add.c` și `sub.c` conțin, respectiv, implementarea funcțiilor `add` și `sub`, iar fișierul `func.h` deține antetele acelor funcții. Pentru obținerea unui executabil, toate aceste fișiere vor fi transmise ca argument compilatorului:

```
$ gcc add.c main.c sub.c
$ ./a.out
Suma (10) = 55
Suma alternanta (10) = -5
```

Se poate folosi opțiunea `-o` pentru a specifica numele executabilului final:

```
$ gcc add.c main.c sub.c -o exec
$ ./exec
Suma (10) = 55
Suma alternanta (10) = -5
```

10.2.2 Optimizarea compilării

Un compilator translatează codul sursă în **cod mașină**. Codul mașină obținut înseamnă, de fapt, instrucțiuni specifice procesorului sistemului pentru care s-a efectuat compilarea. Drept urmare, compilatorul trebuie să aibă cunoștințe foarte importante despre instrucțiunile și registrele procesorului. Din această cauză, partea de translatare a codului sursă are, de obicei, o fază de **optimizare a codului**.

Denumirea de optimizare este ușor forțată. Un termen mult mai bun este cel de „îmbunătățire"; nu se poate spune că se obține „cod optim".

Optimizarea are drept scop obținerea unui executabil care să ruleze cât mai rapid și/sau care să ocupe cât mai puțin spațiu.

Se poate întâmpla ca aceste două scopuri: viteză mare, spațiu puțin, să fie conflictuale; utilizatorul va putea preciza compilatorului care scop este mai important. O dată cu dezvoltarea sistemelor embedded, un rol important pe care trebuie să-l îndeplinească executabilul obținut de compilator este consumul redus de putere. Optimizarea codului are dezavantajului unui timp mai mare de compilare. Din acest motiv, partea de dezvoltare a unei aplicații se realizează fără

optimizare. Faza de optimizare este folosită în momentul în care aplicaţia este pregătită pentru lansare (*release*).

GCC permite optimizarea codului obţinut prin intermediul opţiunii `-O`. Aceasta este urmată de un număr care specifică gradul de "optimizare"/îmbunătăţire a codului. În mod implicit, compilatorul nu optimizează codul. Acest lucru este echivalent cu folosirea opţiunii `-O0` (litera „o" mare şi zero):

```
$ gcc -c chatserv.c -O0
$ ls -l chatserv.o
-rw-r--r-- 1 ubuntu ubuntu 3656 Sep 18 23:46 chatserv.o
```

GCC prezintă 3 niveluri numerice de compilare:

- opţiunea `-O1` înseamnă reducerea timpului de execuţie şi a codului obţinut fără a afecta semnificativ timpul de compilare; este echivalentă cu folosirea opţiunii `-O` (fără un argument numeric)

```
$ gcc -c chatserv.c -O1
$ ls -l chatserv.o
-rw-r--r-- 1 ubuntu ubuntu 3200 Sep 18 23:46 chatserv.o
```

- opţiunea `-O2` înseamnă folosirea tuturor optimizărilor care nu atrag un compromis viteza-spaţiu

```
$ gcc -c chatserv.c -O2
$ ls -l chatserv.o
-rw-r--r-- 1 ubuntu ubuntu 3232 Sep 18 23:47 chatserv.o
```

Se observă că s-a obtinut un cod ceva mai mare decât în cazul opţiunii `-O1`, dar care va rula mai rapid.

- opţiunea `-O3` este în acest moment cea mai puternică (şi agresivă) metodă de compilare

```
$ gcc -c chatserv.c -O3
$ ls -l chatserv.o
-rw-r--r-- 1 ubuntu ubuntu 3308 Sep 18 23:47 chatserv.o
```

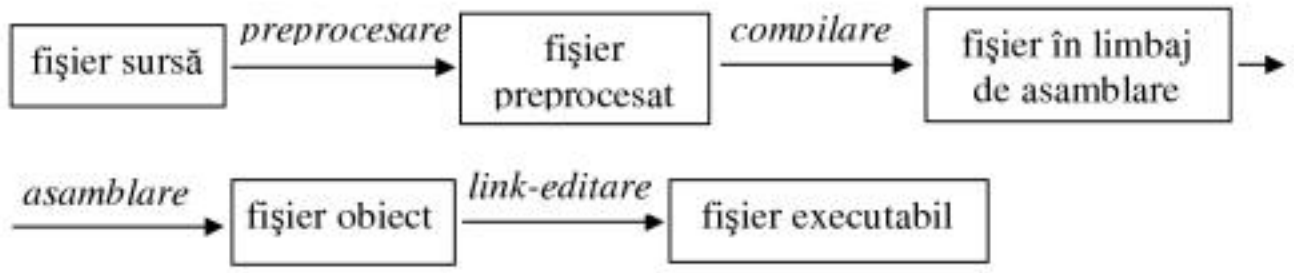
GCC oferă şi opţiunea `-Os` care înseamnă optimizarea codului pentru a ocupa cât mai puţin spaţiu. O utilizare tipică este în cazul dispozitivelor embedded care nu dispun de resurse hardware importante:

```
$ gcc -c chatserv.c -Os
$ ls -l chatserv.o
-rw-r--r-- 1 ubuntu ubuntu 2868 Sep 18 23:47 chatserv.o
```

10.2.3 Etapele compilării (inclusiv link-editare)

Exemplele de până acum au folosit GCC pentru a obţine dintr-un fişier sau mai multe fişiere sursă un executabil. Acest lucru a fost realizat într-o singură comandă. Totuşi, rularea unei instanţe a GCC presupune trecerea printr-un set de etape de prelucrare a fişierului sursă până la executabil. Aceste faze, prezente în continuare, sunt, în ordine, preprocesare, compilare, asamblare şi link-editare. O rulare simplă a GCC înseamnă trecerea prin toate aceste faze. Prezenţa unor parametri poate forţa GCC să se oprească după una dintre ele.

Imaginea de mai jos sintetizează etapele compilării:



10.2.3.1 Preprocesarea

Etapa de preprocesare presupune înlocuirea directivelor de preprocesare din fişierul sursă C. Directivele de preprocesare încep cu #.

Printre exemple se numără directiva `#include` şi directiva `#define`. După preprocesare, directiva `#include` este înlocuită cu fişierul inclus, iar directiva `#define` defineşte un macro care va fi substituit cu valoarea sa oriunde apare în codul sursă.

În exemplul de mai jos se realizează preprocesarea fişierului `info.c`:

```
$ cat info.h
#define MAX_ARRAY_LENGTH        32

struct array_strunct {
        int array[MAX_ARRAY_LENGTH];
        int len;
};
$ cat info.c
#include "info.h"

int get_sum (struct array_struct *a)
{
        int i;
        int sum;

        if (a->len >= MAX_ARRAY_LENGTH)
                return 0;

        for (i = 0; i < a->len; i++)
                sum += a->array[i];

        return sum;
}
$ gcc -E info.c
# 1 "info.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "info.c"
# 1 "info.h" 1


struct array_strunct {
 int array[32];
 int len;
};
# 2 "info.c" 2

int get_sum (struct array_struct *a)
{
 int i;
 int sum;

 if (a->len >= 32)
  return 0;

 for (i = 0; i < a->len; i++)
  sum += a->array[i];

 return sum;
}
```

Se observă înlocuirea directivei `#include` cu fişierul `info.h` şi înlocuirea macrodefiniţiei `MAX_ARRAY_LENGTH` cu `7` în fişierul preprocesat.

Pentru oprirea rulării GCC după faza de preprocesare se foloseşte opţiunea `-E`. Precizarea fişierului de ieşire se realizează cu ajutorul opţiunii `-o`:

```
$ gcc -E info.c -o info.i
```

Preprocesarea este efectuată pe un sistem Linux cu ajutorul preprocesorului **GNU CPP (C preprocessor)**.

10.2.3.2 Compilarea

Compilarea este etapa în care din fişierul preprocesat se obţine un fişier în limbaj de asamblare.

Compilarea efectivă înseamnă aşadar, traducerea codului din limbaj de nivel înalt în limbaj de asamblare care foloseşte instrucţiunile specifice procesorului sistemului fizic. Pentru oprirea rulării GCC după faza de compilare se foloseşte opţiunea `-S`. La fel, se poate preciza fişierul de ieşire cu ajutorul opţiunii `-o`. În continuare se compilează fişierul `info.c` din secţiunea anterioară:

```
$ gcc -S info.c

$ cat info.s
        .file   "info.c"
        .text
.globl get_sum
        .type   get_sum, @function
get_sum:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $20, %esp
        movl    8(%ebp), %eax
        movl    128(%eax), %eax
        cmpl    $31, %eax
        jle     .L2
        movl    $0, -20(%ebp)
        jmp     .L4
.L2:
        movl    $0, -8(%ebp)
        jmp     .L5
.L6:
        movl    -8(%ebp), %eax
[…]
```

Se recomandă ca fişierul în limbaj de asamblare obţinut să aibă extensia `.s`.

Pentru a justifica efectul compilării optimizate vom compila următorul fişier atât cu opţiune de optimizare, cât şi fără:

```
$ gcc -S info.c -O2
$ cat info.s
        .file   "info.c"
        .text
        .p2align 4,,15
.globl get_sum
        .type   get_sum, @function
get_sum:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        movl    8(%ebp), %ebx
        movl    128(%ebx), %ecx
        cmpl    $31, %ecx
        jg      .L10
        testl   %ecx, %ecx
        jle     .L4
        xorl    %edx, %edx
        .p2align 4,,15
.L6:
        addl    (%ebx,%edx,4), %eax
[...]
```

Se observă, în cazul compilării optimizate, că s-au obținut mult mai puține instrucțiuni în limbaj de asamblare şi, deci, un cod care ocupă mai puțin spațiu şi se execută mai rapid.

10.2.3.3 Limbaje de asamblare

Limbajul de asamblare este un limbaj de programre de nivel scăzut. Instrucțiunile sale sunt de fapt reprezentare simbolică a instrucțiunilor puse la dispoziție de procesorul sistemul.

Astfel, un program scris în limbaj de asamblare este neportabil, în sensul că poate fi rulat numai pe procesorul pentru care a fost scris. Limbajul de asamblare oferă utilizatorului mnemonici pentru reprezentarea instrucțiunilor procesorului: `mov`, `cmp`, `add`, `sub`, `push`, etc. Astfel instrucțiunea de procesor `10110000 01100001` (`0xb061`) este scrisă în limbaj de asamblare ca `mov al, 061h`. Mnemonica `mov` înseamnă efectuarea unei copieri. Rezultatul este încărcarea registrului `al`, al procesorului, cu valoarea `0x61`. Există, în general, o asociere unu la unu între o instrucțiune în limbaj de asamblare şi o instrucțiune în cod maşină.

Programarea în limbaj de asamblare necesită cunoştințe despre structura procesorului, resursele acestuia, mecanismul de acces la memorie. Din această cauză, programarea în limbaj de asamblare este dificilă şi folosită doar atunci când este nevoie: când un limbaj cum este C nu oferă instrucțiuni pentru efectuarea unei anumite operații sau când este nevoie de eficiență maximă care poate fi obținută numai prin scrierea de cod în limbaj de asamblare.

Traducerea codului în limbaj de asamblare în cod obiect se realizează cu ajutorul unui asamblor în faza de asamblare descrisă în secțiunea următor. Fiind vorba de o asociere unu la unu, este posibilă obținerea codului în limbaj de asamblare pornind de la codul obiect. Această operație poartă numele de dezasamblare şi este prezentată în secțiunea 10.2.3.5.

Tipuri de limbaj de asamblare

Chiar pentru acelaşi procesor pot exista limbaje de asamblare diferite. Mai corect este expresia sintaxe de limbaj de asamblare diferite. Pe arhitectura x86, asamblorul de la Microsoft foloseşte sintaxa Intel, pe când asamblorul de Linux (GNU) foloseşte sintaxa AT&T. Astfel, copierea valorii `0x100` în registrul `eax` se face, în cele două limbaje de asamblare, în modul prezentat mai jos:

```
movl   $0x100, %eax
mov    eax, 100h
```

La fel, stocarea valorii `100` la adresa indicată de `eax`:

```
movl $0x100, (eax)
movl [eax], 100h
```

10.2.3.4 Asamblarea

Etapa de asamblare este etapa de traducere a codului scris în limbaj de asamblare (fişier text conţinând mnemonici specifice procesorului) în cod binar.

Acest cod binar este de fapt cod maşină reprezentând codificarea binară a instrucţiunilor procesorului. Fişierul obţinut poartă numele de fişier **cod obiect** sau **modul obiect**. Pe Unix un modul obiect are extensia `.o` iar pe Windows extensia `.obj`.

Pentru oprirea rulării `gcc` după faza de asamblare se foloseşte opţiunea `-c`. În mod implicit se înlocuieşte extensia fişierului sursă cu `.o`. Fişierul sursă poate fi un fişier C sau un fişier în limbaj de asamblare:

```
$ gcc -c info.c
$ ls -l info.o
-rw-r--r-- 1 ubuntu ubuntu 764 Sep 19 00:29 info.o
$ gcc -c info.s
$ ls -l info.o
-rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:29 info.o
```

Ca şi până acum se poate folosi opţiunea -o pentru a schimba numele implicit al fişierului de ieşire:

```
$ gcc -c info.s -o modul.o
$ ls -l modul.o
-rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:47 modul.o
```

În suita de aplicaţii GNU, asamblorul poartă numele de GAS (*GNU Assembler*), iar executabilul folosit este `as`. Un fişier în limbaj de asamblare poate fi asamblat folosind `as`:

```
$ as info.s -o as_out.o
$ ls -l as_out.o
-rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:48 as_out.o
```

10.2.3.5 Codul obiect

Un modul obiect conţine cod maşină specific procesorului. Acest cod este obţinut în urma fazelor de preprocesare, compilare şi asamblare. Un modul obiect are, de obicei, extensia `.o`:

```
$ file info.o
info.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not
stripped
```

objdump; dezasamblare

Din cauza asocierii 1 la 1 care există între codul obiect şi codul în limbaj de asamblare, se poate realiza o operaţie de dezasamblare. Utilitarul folosit în acest sens este `objdump`:

```
$ objdump -d info.o
info.o:     file format elf32-i386

Disassembly of section .text:

00000000 <get_sum>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 14                sub    $0x14,%esp
   6:   8b 45 08                mov    0x8(%ebp),%eax
   9:   8b 80 80 00 00 00       mov    0x80(%eax),%eax
   f:   83 f8 1f                cmp    $0x1f,%eax
  12:   7e 09                   jle    1d <get_sum+0x1d>
  14:   c7 45 ec 00 00 00 00    movl   $0x0,0xffffffec(%ebp)
[...]
```

nm

Un alt utilitar care este folositor în lucrul cu fişiere obiect este `nm`. Cu ajutorul acestuia se pot lista simbolurile dintr-un fişier obiect:

```
$ nm info.o
00000000 T get_sum
```

nm poate oferi informaţii despre definirea sau nu a unui simbol într-un modul obiect:

```
razvan@anaconda:~/junk/compilare$ nm main.o
         U add
00000000 T main
         U printf
         U sub
```

Din rezultatul afişării se observă că simbolurile sunt marcate cu U sau T. U înseamnă că acel simbol este nedefinit, iar T înseamnă că este nedefinit în modulul obiect primit ca argument. În exemplul nostru, modulul are definit simbolul `main` dar nedefinit simbolurile `add`, sub şi `printf`. Aceste simboluri se găsesc în alte module obiect sau în biblioteci. Rezolvarea acestor simboluri se va realiza în momentul link-editării.

10.2.3.6 Link-editarea

Un modul obiect conţine doar traducerea codului în limbaj de asamblare în cod maşină specific procesorului pe care lucrează.

Pentru obţinerea unui fişier executabil şi deci pentru rularea acestui cod este necesară „legarea" diverselor simboluri. Această operaţie se numeşte link-editare, linking sau legare.

În momentul în care se compilează un modul obiect care foloseşte funcţia de bibliotecă `printf`, aceasta apare ca simbol nedefinit:

```
razvan@anaconda:~/junk/compilare$ nm main.o
00000000 T main
         U printf
```

Acest simbol este definit în biblioteca standard C. În urma link-editării se obţine un executabil care importă acel simbol din biblioteca standard C. Altă situaţie este folosirea într-un modul obiect a unei funcţii definite într-un alt modul obiect; un exemplu este folosirea funcţiei `add` definită în `add.c` în fişierul `main.c` în secţiunea 10.2.1.1. Aplicaţia care realizează acest lucru se numeşte linker. Rolul unui linker este, aşadar, de a lega mai multe fişiere obiect sau biblioteci, de a rezolva simbolurile nedefinite şi de a obţine un executabil sau o bibliotecă. Figura de mai jos explică rolul tipic pe care îl îndeplineşte un linker:

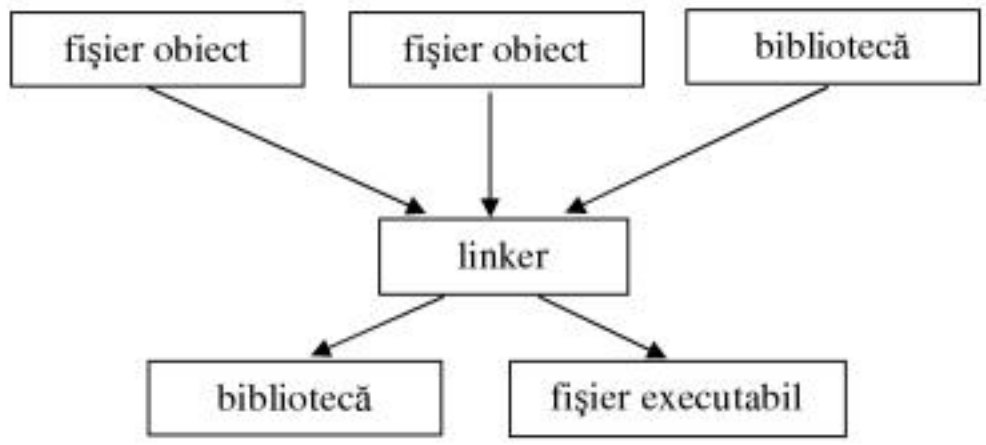


Figura 10-1 Funcţionarea unui linker

Mai multe detalii despre biblioteci se găsesc în secţiunea 10.3.

Invocare linker

Linker-ul de pe distribuţiile Linux este **GNU LD**. Executabilul asociat este `ld`. LD este rareori invocat de unul singur. În mod normal, el este folosit de GCC:

```
$ gcc -Wall -c main.c
$ gcc -Wall -c add.c
$ gcc -Wall -c sub.c
$ gcc main.o add.o sub.o -o exec
$ ./exec
Suma (10): 55
Suma alternanta (10): -5
```

Se poate folosi şi `ld`, însă numărul mare de argumente care trebuie transmis face această opţiune mai puţin viabilă:

```
  $ ld -dynamic-linker /lib/ld-linux.so.2 /usr/lib/crt1.o /usr/lib/crti.o
/usr/lib/gcc/i486-linux-gnu/4.1.2/crtbegin.o main.o add.o sub.o -lc
/usr/lib/gcc/i486-linux-gnu/4.1.2/crtend.o /usr/lib/crtn.o -o exec
  $ ./exec
  Suma (10): 55
  Suma alternanta (10): -5
```

10.2.3.7 Fişiere executabile

Un **fişier executabil** este un fişier binar obţinut dintr-un set de fişiere obiect. Spre deosebire de fişierele obiect, un fişier executabil are identificate şi eventual rezolvate simbolurile. Aceste simboluri sunt rezolvate la linking (legare statică) sau la rulare (legare dinamică). În momentul execuţiei, informaţiile conţinute în fişierul executabil sunt folosite pentru a genera un proces. Un fişier executabil va conţine codul maşină folosit pentru îndeplinirea sarcinilor date dar şi antete şi secţiuni de formatare sau descriere auxiliare. Acestea sunt folosite pentru a descrie modul în care se foloseşte codul, zonele de memorie folosite, etc. Un fişier executabil are, aşadar, un format bine definit. Formatul de fişier executabil se referă la fişiere executabile şi la biblioteci.

Formate ale fişierelor executabile

Formatul unui fişier executabil este strâns legat de sistemul de operare. Sistemul de operare este responsabil cu interpretarea unui fişier executabil şi generarea unui proces pe baza acestuia. În Windows, formatul se numeşte `PE` (**Portable Executable**)[17].

În Linux, formatul se numeşte `ELF` (**Executable and Linking Format**)[18]:

```
  $ file exec
  exec: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for
GNU/Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux 2.4.1,
not stripped
  $ file /bin/ls
  /bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for
GNU/Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux 2.4.1,
stripped
  $ file /lib/libc-2.3.6.so
  /lib/libc-2.3.6.so: ELF 32-bit LSB shared object, Intel 80386, version 1
(SYSV), for GNU/Linux 2.4.1, stripped
```

Utilitarul `readelf` poate fi folosit pentru analiza unui fişier în format ELF:

```
razvan@anaconda:~/junk/compilare$ readelf -S exec
There are 34 section headers, starting at offset 0xe38:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048114 000114 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048128 000128 000020 00   A  0   0  4
  [ 3] .hash             HASH            08048148 000148 000028 04   A  4   0  4
[...]
  [10] .init             PROGBITS        08048258 000258 000017 00  AX  0   0  4
  [11] .plt              PROGBITS        08048270 000270 000040 04  AX  0   0  4
  [12] .text             PROGBITS        080482b0 0002b0 0001c4 00  AX  0   0 16
  [13] .fini             PROGBITS        08048474 000474 00001c 00  AX  0   0  4
  [14] .rodata           PROGBITS        08048490 000490 00001a 00   A  0   0  4
```

În exemplul de mai sus, s-au afişat secţiunile unui fişier executabil în format ELF.

10.3 Biblioteci de funcţii

Bibliotecile de funcţii (engleză: *libraries*) au aparut pentru a oferi o mai mare flexibilitate dezvoltatorilor de software. Acestea reunesc funcţii des utilizate de către mai multe programe, fără a fi necesară rescrierea acestora pentru fiecare program în parte.

În funcţie de utilizarea bibliotecilor, ele se pot împărţi astfel:

- **bibliotecile standard** (`libc`, `libstdc++`) – oferă o modalitate de acces unitar la funcţiile puse la dispoziţie de kernel sau oferă implementări pentru diferite structuri de date utilizate în programe.
- **bibliotecile de desenare a interfeţelor grafice** (`GTK+`, `Qt`) – oferă un aspect unitar (butoane desenate la fel etc.) aplicaţiilor care doresc să facă uz de interfaţa grafică în interacţiunea cu utilizatorul.
- **bibliotecile de procesare a imaginilor** (`libjpeg`, `libpng`) – oferă un set comun de funcţii pentru procesarea unor formate de imagini.
- **biblioteci utilitare** – înglobează funcţii utilizabile de către mai multe programe (biblioteci care implementează obiecte pentru accesul la reţea etc.)

---

17 ***http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx***

18 *http://x86.ddj.com/ftp/manuals/tools/elf.pdf*

Acestea sunt doar câteva dintre aplicaţiile bibliotecilor. Practic, orice funcţionalitate utilizabilă de mai multe aplicaţii poate fi implementată într-o bibliotecă.

10.3.1 Tipuri de biblioteci

Exista trei tipuri de biblioteci de funcţii:

- biblioteci statice
- biblioteci partajate
- biblioteci dinamice

Toate bibliotecile de funcţii există sub formă de fişiere independente în sistem.

Bibliotecile statice (static libraries) sunt colecţii de funcţii existente sub forma de fişiere, care la link-editare sunt incluse complet în executabilele obţinute.

Bibliotecile partajate (shared libraries) sunt colecţii de funcţii existente sub forma de fişiere care, în momentul link-editării, sunt referite (reffered) în codul executabil rezultat şi încărcate de către sistem în momentul apelării programului.

Bibliotecile partajate reprezintă metoda cea mai utilizată de a pune la dispoziţie funcţii comune pentru mai multe aplicaţii.

Bibliotecile dinamice (dynamic libraries) sunt colecţii de funcţii existente sub formă de fişiere, colecţii care pot fi încărcate la cerere de către programe în timpul execuţiei.

Ca exemplu de utilizare pentru bibliotecile dinamice se poate aminti programarea plugin-urilor (extensie pentru un program) pentru extinderea aplicaţiilor.

10.3.2 Informaţii despre bibliotecile de funcţii

Bibliotecile statice, dat fiind utilitatea lor doar în procesul de compilare, se găsesc în general în `/usr/lib` sau în `/usr/local/lib` şi au extensia `.a` (provine de la `archive`)

Bibliotecile partajate, având în vedere că sunt utile permanent în sistem, se găsesc în general în directorul `/lib` şi au extensia `.so` (provine de la `shared object` – obiect partajat).

Bibliotecile dinamice au acelaşi format (din punct de vedere al structurii) ca şi bibliotecile partajate, singura diferenţă fiind modul de încărcare. Ele au tot extensia `.so`, dar sunt încărcate explicit de către aplicaţii, localizarea lor fiind făcută de aplicaţie.

Informaţii primare despre bibliotecile de funcţii se pot afla prin intermediul comenzii `file`. Mai jos este prezentată ieşirea pentru comanda `file`, atât pentru o bibliotecă statică cât şi pentru o bibliotecă dinamică.

În cazul unei biblioteci statice:

```
$ file /usr/lib/libm.a
/usr/lib/libm.a: current ar archive
```

Bibliotecile statice sunt de fapt arhive de fişiere obiect. Ele sunt create prin intermediul comenzii `ar`. (pentru mai multe informaţii se poate consulta `man ar`):

În directorul `/lib` se găsesc foarte multe fişiere cu extensia `.so` urmată de o serie de numere. Acesta este modul în care se realizează diferenţierea versiunilor.

```
$ file /lib/libm.so.6
/lib/libm.so.6: symbolic link to `libm-2.3.3.so'

$ file /lib/libm-2.3.3.so
/lib/libm-2.3.3.so: ELF 32-bit LSB shared object, Intel 80386, version 1
(SYSV), not stripped
```

Observaţie: `shared object` denotă faptul că este o bibliotecă partajată

După cum a fost prezentat şi anterior, comanda `nm` poate fi utilizată pentru a vizualiza lista simbolurilor dintr-o bibliotecă (statică sau dinamică). Prin simboluri se înţeleg denumirile date funcţiilor publice (funcţii ce pot fi utilizate de programe) din aceste biblioteci. Exemplu:

```
$ nm /lib/libm.so.6
00052b60 T fscanf
00043f20 T fprintf
00052b90 T scanf
00052bd0 T sscanf
         U __libc_enable_secure@@GLIBC_PRIVATE
         U __libc_stack_end@@GLIBC_2.1
```

Simbolurile care sunt precedate de caracterul `U` sunt simboluri care trebuie rezolvate de linker la apelarea programului (se observă că ele nu au adresă în fişierul cu biblioteca). Simbolurile acestea se regăsesc în alte biblioteci. Simbolurile marcate cu `T` sunt simboluri din zona de cod (funcţii) ce pot fi folosite extern de către programe sau alte biblioteci.

10.4 Automatizarea sarcinilor – make

`Make` este un program utilizat pentru automatizarea task-urilor. El rezolvă problema execuţiei unor acţiuni în funcţie de relaţiile (dependeţele) între ele. Astfel, o acţiune este executată doar dacă acţiunile de care depinde ea au fost executate. Relaţiile între acţiuni sunt definite într-un fişier numit `Makefile`. Pe baza acestuia, `make` determină automat secvenţa de paşi care trebuie efectuată pentru respectarea dependinţelor. Pentru a optimiza execuţia acţiunilor, `make` determină dacă o acţiune a fost efectuată şi nu o mai execută. În acelaşi timp, `make` detectează dacă pentru o acţiune s-a modificat o dependinţă şi o execută din nou.

Cele mai întâlnite utilizări pentru `make` sunt:

- recompilarea programelor compuse din mai multe fişiere
- testarea programelor.

10.4.1 Terminologie

Instrucţiunile pentru make se găsesc într-un fişier cu numele Makefile/makefile. Ele determină ce acţiuni trebuie executate pentru a satisface anumite cerinţe. Sintaxa unui fişier `Makefile` este de forma:

```
$ cat Makefile
Target... : dependinţe ...
<tab>comandă
<tab>comandă
<tab>...
```

unde:

- `target` = numele unui fişier care trebuie generat sau numele unei acţiuni; în cazul în care target-ul este un fişier, comenzile trebuie să genereze acest fişier; în cazul în care comenzile nu generează un fişier cu numele target-ului, se va considera la fiecare rulare ca target-ul nu a fost creat (aşadar toate comenzile se vor executa).
- `dependinţe` = lista de target-uri (separate prin spaţii) care trebuie îndeplinite (fişiere care trebuie să existe) pentru a se realiza target-ul curent – `make` poate determina dacă trebuie reexecutat target-ul dacă una dintre dependeţe s-a modificat (unul din fişierele din lista de dependinţe s-a modificat)

- `comandă` = comanda care duce la realizarea target-ului (de obicei la comenzi sunt trecute comenzile de compilare care duc la realizarea unui fişier cu numele target-ului)
- `<tab>` = caracterul Tab

Regulile formează un lanţ de dependinţe. Pentru că un target să fie executat, se verifică dacă toate dependinţele lui există ca fişiere. Dacă nu este îndeplinită această condiţie, dependiţele care nu există sunt executate, aplicându-se acelaşi algoritm.

Pentru ca, în momentul execuţiei, comanda să nu fie afişată pe ecran, aceasta trebuie prefixată cu semnul `@`.

```
$ cat Makefile
.PHONY: all
all: dep1 dep2
        ...
        @echo "Am executat tot"
```

Există cazuri în care vrem să executam un target chiar dacă el a fost deja executat; spre exemplu, în referirea de mai sus se doreşte ca target-ul cu numele `all` să se execute de fiecare dată când este apelat, chiar dacă există un fişier cu numele „`all`".

Pentru a realiza acest lucru, acest target trebuie marcat cu `.PHONY` (în traducere „fals"). Se recomandă marcarea tuturor target-urilor care nu produc un fişier cu numele target-ului cu atributul `.PHONY`, precum s-a văzut anterior. Acest lucru previne eventuala situaţie când target-ul nu s-ar executa deoarece `make` vede că există un fişier cu acelaşi nume şi consideră că target-ul s-a executat deja.

10.4.2 Mod de utilizare

La execuţia comenzii `make`, vor fi luate în considerare target-urile definite în fişierul `Makefile` din directorul curent.

Pentru a executa un target, se foloseşte sintaxa:

```
$ make target
```

unde `target` este numele target-ului care se doreşte a fi executat.

La utilizarea lui *`make`* fără parametrii, se va executa primul target definit în fişierul `Makefile`. Din acest motiv, primul target descris este de obicei cel mai complex, cu cele mai multe dependinţe.

Pentru compilarea unui program C, un fişier `Makefile` simplu ar arăta în felul următor:

```
$ cat Makefile
.PHONY: all
all: progc

progc: m.c
        gcc m.c -Wall -o progc

.PHONY: clean
clean:
        rm progc
```

Trebuie observat că s-a folosit numele executabilului obţinut în urma compilării ca nume pentru target. Acest lucru permite comenzii *`make`* să detecteze când un target a fost făcut deja şi să îl sară în eventualitatea în care este prezent ca o dependinţă.

Pentru o flexibilitate mai mare în realizarea target-urilor, se pot introduce variabile în interiorul unui `Makefile`:

```
$ cat Makefile
gccopt=-Wall
optimizare=-O2
ccompiler=gcc

.PHONY: all
all: progc

progc:
        $(ccompiler) m.c $(gccopt) $(optimizare) -o progc

.PHONY: clean
clean:
        rm progc
```

Se observă că:

- s-a scos opțiunea `-Wall` într-o variabilă separată
- s-a adăugat opțiunea `optimizare` tot ca o variabilă
- compilatorul a fost trecut într-o variabilă separată

Principalul avantaj pe care îl aduce utilizarea variabilelor este flexibilitatea crescută în realizarea şi modificarea comenzilor de compilare pentru diferite condiții.

Programele care vin ca surse se distribuie în general cu un script de configurare, numit `configure`. Acesta are ca scop stabilirea unor variabile ce sunt ulterior folosite de un `Makefile`. Aceste variabile pot configura diferite lucruri, precum compilatorul care va fi folosit, bibliotecile care vor fi incluse, etc.

`make` poate funcționa şi fără existența unui `Makefile`:

- pentru a compila direct fişierul `myprog.c` (care nu depinde de altele)

```
$ make myprog
```

- pentru a compila direct `myprog.c` cu parametrul de compilare `-Wall`

```
$ make CFLAGS=-Wall myprog
```

10.5 Portabilitate

Portabilitatea este caracteristica unei aplicații de a putea fi folosită într-un alt mediu în afara aceluia pentru care a fost inițial proiectată. Acest lucru poate însemna un alt sistem de operare, o altă arhitectură hardware, o altă bibliotecă. Se spune că o aplicație este portabilă dacă are nevoie de modificări minime pentru a putea rula pe un alt mediu. Acțiunea de portare este acțiunea de modificare a unei aplicații pentru a putea fi folosită pe un alt sistem de operare sau sistem fizic. Un mediu pe care se realizează portarea se mai numeşte platformă.

Dacă în deceniile trecute numărul de arhitecturi posibile pe care rula o aplicație era destul de mare, în zilele noastre arhitectura hardware dominantă este x86. La fel sistemele de operare pe care se portează o aplicație sunt Microsoft Windows, Apple Mac OS X şi Unix/Linux. În lumea sistemelor embedded, însă, portabilitatea rămâne o problemă importantă.

10.5.1 Portabilitatea la nivelul arhitecturii hardware (procesorului)

După cum s-a precizat, în sistemele desktop arhitectura dominantă este x86 aşa că rareori este nevoie de portarea unei aplicații. În sistemele embedded portarea la nivelul procesorului este un factor important. Totuşi portarea la nivelul procesorului poate să nu fie necesară datorită compilatorului. Astfel, pentru un limbaj de nivel înalt, compilatorul va asigura traducerea

codului sursă în cod maşină specific procesorului dat. GCC este cel mai portat şi cel mai portabil compilator fiind capabil de a compila programe pentru o varietate de arhitecturi.

10.5.2 Portabilitatea unui limbaj de programare

Programarea în limbaj de asamblare nu asigură, prin definiţie, portabilitatea codului, deorece acesta este intrinsec legat de arhitectura sistemului de calcul. Unul dintre motivele răspândirii Unix în anii '70 a fost scrierea acestuia în C, un limbaj de nivel înalt portabil. Folosirea C însemna eliberarea de arhitectura hardware pe care va rula aplicaţia. Portarea codului era făcută prin intermediul compilatorului. În ziua de azi, aplicaţiile C sunt cele mai uşor portabile datorită existenţei unui compilator C pe orice platformă.

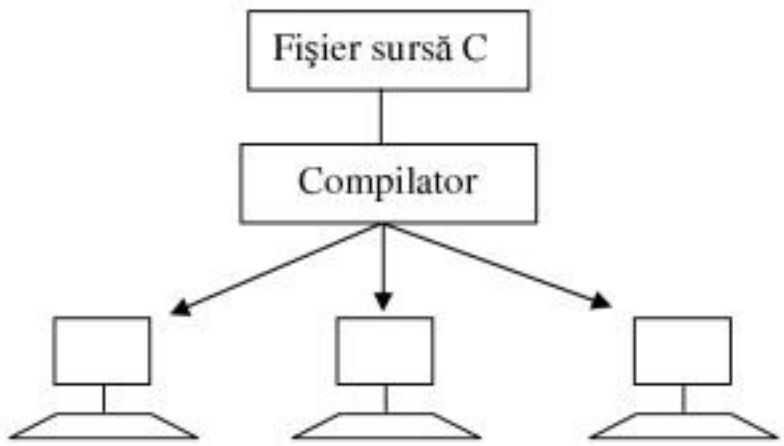


Figura 10-2 Portabilitatea asigurată de compilator

Java este un limbaj de programare gândit pentru a fi portabil. Deviza Java este *"Write once, run everywhere"*. Pentru a asigura acest lucru, un program Java este compilat într-o formă portabilă de modul obiect denumită **bytecode**. Acest bytecode este apoi interpretat de o maşină virtuală Java. Maşina virtuală Java este responsabilă cu translatarea bytecode-ului portabil în cod specific sistemului pe care acesta rulează. Astfel, rularea unui program Java sau unui modul compilat în bytecode pe o nouă platformă este condiţionată de existenţa unei maşini virtuale pe acea platformă. Maşina virtuală Java este conţinută în pachetul `JRE` de la Sun (**Java Runtime Environment**). Acesta conţine maşina virtuală şi bibliotecile cu funcţii standard Java.

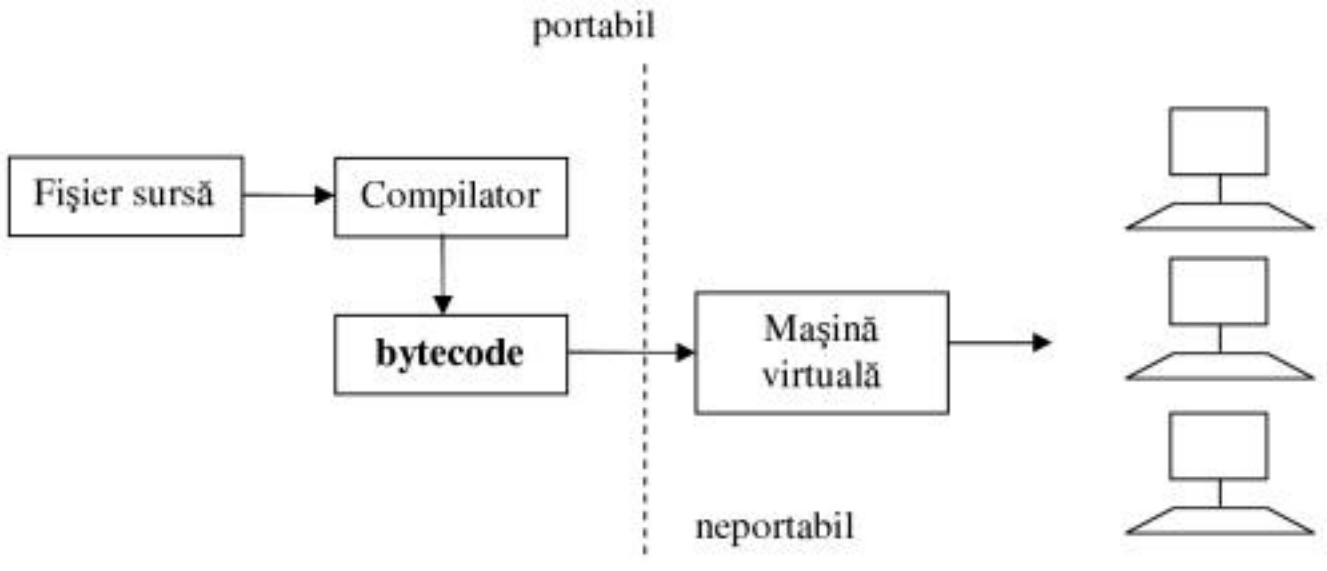


Figura 10-3 Portabilitatea asigurată de maşina virtuală

10.5.3 Portabilitatea la nivelul sistemului de operare

Portabilitatea unei aplicații se poate referi și la posibilitatea rulării acesteia pe un alt sistem de operare. Acest lucru este corelat, de obicei, cu interfețele de programare (`API` - *Application Programming Interface*) pe care sistemul de operare le pune la dispoziție (*system API*). Astfel, Windows pune la dispoziția programatorului C interfața `Win32API`. Unix, pe de altă parte pune la dispoziția programatorului interfața `POSIX` (Portable Operating System Interface). Pentru deschiderea unui fișier un programator Windows va folosi `CreateFile`, iar un programator Unix `open`.

Pentru asigurarea portabilității însă, multe biblioteci oferă programatorului o interfață portabilă peste sistemul de operare. Astfel, funcțiile ANSI din biblioteca standard C sunt portabile peste diverse sisteme de operare. Astfel, pentru ca deschiderea unui fișier, programatorul va folosi `fopen` indiferent de sistemul de operare pe care va rula aplicația.

Un exemplu de bibliotecă portabilă este `wxWidgets`[19]. Această bibliotecă permite crearea de aplicații grafice peste diferite sisteme de operare prin intermediul unui API comun. Nu numai că acest lucru va ascunde comunicarea cu sistemul de operare din spate dar și cu bibliotecile grafice neportabile pe care acesta le pune în mod obișnuit pentru crearea de primitive grafice.

10.6 Execuția programelor

10.6.1 Detalii despre execuția programelor

În momentul execuției, sistemul de operare efectuează o serie de operații pentru a pregăti aplicația pentru rulare. Procesele lucrează în zone de memorie independente unele de altele. Astfel, ceea ce se găsește la o anumită adresă pentru un proces nu se găsește și la alt proces.

Pe lângă încărcarea zonei de cod și a zonei de date din executabil, sistemul de operare trebuie să încarce și bibliotecile partajate. Bibliotecile statice devin parte din aplicații și nu au cum să fie tratate la execuție.

Pentru a vizualiza lista bibliotecilor partajate utilizate de un program, se folosește comanda `ldd`, precum în exemplul următor:

```
$ ldd /bin/bash
        linux-gate.so.1 =>  (0xffffe000)
        libreadline.so.5 => /lib/libreadline.so.5 (0xb7eb4000)
        libhistory.so.5 => /lib/libhistory.so.5 (0xb7eac000)
        libncurses.so.5 => /lib/libncurses.so.5 (0xb7e6a000)
        libdl.so.2 => /lib/libdl.so.2 (0xb7e66000)
        libc.so.6 => /lib/libc.so.6 (0xb7d3a000)
        /lib/ld-linux.so.2 (0xb7f0d000)
```

Informațiile despre bibliotecile partajate aflate în sistem sunt colectate (manual sau automat) utilizând programul `ldconfig`. Această comandă:

- caută biblioteci partajate în locațiile standard și în directoarele specificate în `/etc/ld.so.conf`,
- configurează legături simbolice corecte în directoarele cu biblioteci (pentru a oferi denumiri standard biblitecilor, din punct de vedere al formatului)
- salvează referințele către biblioteci într-un cache.

Rularea programului `ldconfig` la fiecare bootare este ineficientă – acesta este un motiv în plus pentru utilizarea unui cache.

---

[19] *wxWidgets - http://www.wxwidgets.org/*

Modul în care programele utilizează bibliotecile de funcţii este definit la link-editare. Linker-ul (`ld`) caută bibliotecile partajate în directoarele standard şi în directoarele adăugate în linia de comandă prin intermediul unor parametri speciali (`-rpath dir` şi `-Ldir`).

Într-un sistem bazat pe biblitecile standard GNU C, inclusiv toate sistemele Linux, pornirea unui executabil de tip ELF (formatul standard al executabilelor) determină încarcarea şi rularea unui **program loader**. Pe sistemele Linux, acesta este `/lib/ld-linux.so.X` (unde X este versiunea). Acest **loader** găseste şi încarcă bibliotecile partajate utilizate de către program.

Dacă se doreşte supraîncărcarea unor funcţii dintr-o bibliotecă dar cu păstrarea restului bibliotecii, se pot introduce bibliotecile supraîncarcate în `/etc/ld.so.preload` – aceste biblioteci „preîncarcate" vor avea întâietate relativ la setul standard. Acest fisier pentru preîncărcări este utilizat de obicei pentru patch-uri de urgenţă – distribuţiile nu includ de obicei un astfel de fişier.

10.6.2 Analiza apelurilor de sistem si a semnalelor

10.6.2.1 Apeluri de sistem (system calls)

Pentru a interacţiona cu sistemul de operare (cel care partajează resursele computerului), bibliotecile de funcţii apelează o serie de functii ale kernel-ului numite apeluri de sistem (*system calls*).

În continuare sunt prezentate doar câteva dintre utilizările apelurilor de sistem:

- deschiderea/inchiderea unui fisier (`open, close`)
- citirea/scrierea dintr-un/intr-un fisier (`read, write`)
- deschiderea unui socket (pentru o conexiune TCP/IP)
- executia unui program (`execv`)

Aceste funcţii sunt implementate la nivelul nucleului sistemului de operare deoarece SO-ul face management-ul resurselor. Bibliotecile de functii „îmbracă" aceste apeluri în funcţii/clase/obiecte utilizabile direct de către programatori.

10.6.2.2 strace

Comanda `strace` este utilă pentru urmărirea apelurilor de sistem efectuate la execuţia unui program. Această analiză reprezintă o metodă destul de eficientă în debugging-ul programelor al căror cod sursa nu-l avem.

Pentru a observa ce informaţii poate să ofere `strace`, se va analiza ieşirea programului pentru o aplicaţie simplă (o parte din ieşirea programului a fost eliminată pentru a uşura analiza):

```
$ strace echo "USO"
execve("/bin/echo", ["echo", "USO"], [/* 51 vars */]) = 0
uname({sys="Linux", node="aquarium", ...}) = 0
[...]
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=141679, ...}) = 0
[...]
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220R\1"..., 512)
= 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1363203, ...}) = 0
[...]
close(3)                                = 0
[...]
open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=38399616, ...}) = 0
[...]
close(3)                                = 0
[...]
fstat64(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 1), ...}) = 0
[...]
write(1, "USO\n", 4USO
)                       = 4
[...]
exit_group(0)                           = ?
```

Se poate observa:

- primul apel de sistem (`execve`) indică ce parametri a primit programul: `echo` şi `USO`
- programul a încercat să deschidă fişierul `/etc/ld.so.preload` dar nu a fost găsit (scopul acestui fişier este descris în subcapitolul dedicat bibliotecilor)
- apelul de sistem `open` a returnat valoarea 3 – aceasta reprezintă un descriptor de fişier (`fd - file descriptor`) – descriptorul de fişier este folosit în procesele de citire şi în procesele de scriere (e un fel de pointer la fişier, pointer utilizat de sistemul de operare)
- apelul de sistem `close` are ca singur parametru – descriptorul de fişier care trebuie închis – o data închis, un descriptor de fişier poate fi refolosit
- apelul de sistem `read` prezent mai jos indică o citire din descriptorul de fişier 3
- apelul de sistem `write` are 3 parametrii:
  - `1` = file descriptor-ul unde se scrie: în cazul nostru, dupa cum ştim, 1 este file descriptor-ul pentru stdout (ieşirea standard)
  - `USO\n` = acesta este textul ce trebuie afişat de comanda `echo`
  - `4` = lungimea textului ce trebuie afisat

În acest program s-au putut identifica câteva apeluri de sistem frecvente.

10.7 Studiu de caz

10.7.1 GCC în Windows

Suportul pentru compilatorul GCC în mediul Windows este disponibil prin proiectele:

- **MinGW**, disponibil la http://www.mingw.org/ - este un set de antete de funcţii folosite pentru a compila aplicaţii folosind GNU GCC – aplicaţiile rezultate folosesc bibliotecile disponibile în Windows şi depind de acestea;

- **Cygwin**, disponibil la http://sourceware.org/cygwin/ - este un mediu care oferă aplicaţiilor un strat (layer) de emulare a API-ului Unix, împreună cu o colecţie de unelte care oferă funcţionalitătile din Unix.

Aceste implementări oferă acces la GCC precum în mediile Linux, respectiv parametrii şi modul de apel ai compilatorului sunt aproape identici.

În mediul Windows este posibila utilizarea GCC-ului şi împreuna cu un mediu integrat de dezvoltare - IDE. `Dev-C++` (http://www.bloodshed.net/devcpp.html) este un astfel de IDE care oferă posibilitatea compilării programelor utilizând GCC.

10.7.2 Compilarea unui program din mai multe surse

S-a observat că, o dată cu creşterea complexităţii codului, este mult mai greu de întreţinut fişierul în care acesta se găseşte şi se optează pentru împărţirea codului în mai multe fişiere. De asemenea, din punct de vedere funcţional, este mult mai simplu de înteles un proiect care are componentele bine definite în fişiere individuale. Compilarea programelor din mai multe surse ridică o serie de probleme ce vor fi discutate în continuare.

Se va porni de la un exemplu foarte simplu de compilare din mai multe fişiere. Fişierele implicate sunt `functii.h`, `functii.c`, `afis.h`, `afis.c`, `myprog.c`:

```
$ cat functii.h
#ifndef _HEADER_FUNCTII
#define _HEADER_FUNCTII

extern int mprod(int a, int b);
extern int mdiv(int a, int b);

#endif
```

```
$ cat functii.c
#include "functii.h"

int mprod(int a, int b) { return a*b; }
int mdiv(int a, int b){ return ((b!=0)?a/b:a); }
```

```
$ cat afis.h
#ifndef _HEADER_AFIS
#define _HEADER_AFIS

extern void afis(char *);
extern void endl();

#endif
```

```
$ cat afis.c
#include "afis.h"
#include <stdio.h>

void afis(char *s){ printf("Text afisat: %s\n",s);  }
void endl(){ printf("\n"); }
```

```
$ cat myprog.c
#include "afis.h"
#include "functii.h"

int main(){
        afis("hello there");
        return (mprod(10,2));
}
```

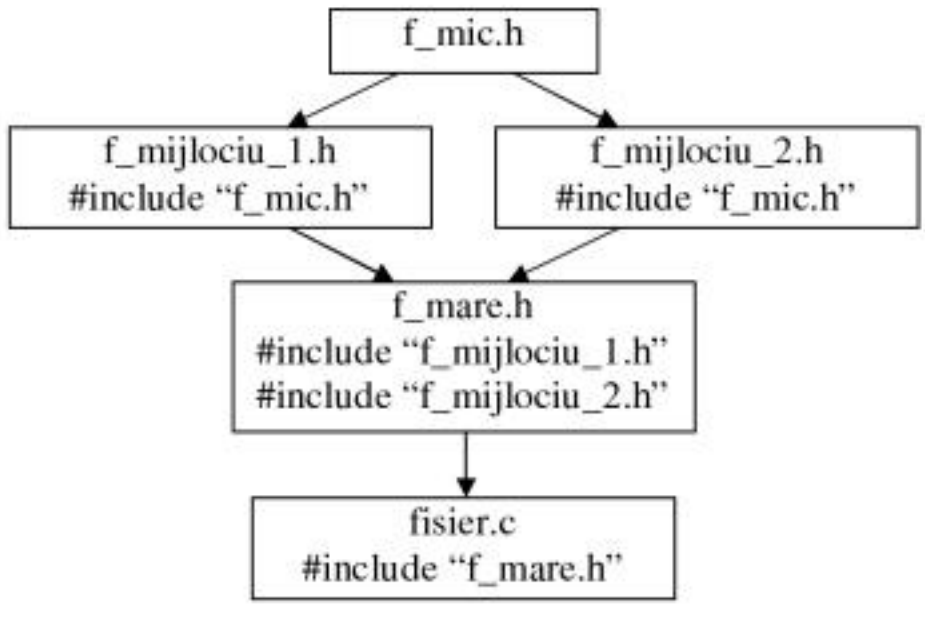
Se doreşte obţinerea unui fişier obiect `functii.o` şi a unui fişier obiect `afis.o`, fiecare dintre ele implementând o funcţionalitate diferită. Aceste două fişiere vor fi link-editate ulterior în programul principal. `Makefile`-ul care realizează compilarea şi link-editarea programului este prezentat în continuare.

```
$ cat Makefile
hello: myprog.o afis.o functii.o
        gcc myprog.o afis.o functii.o -o hello

functii.o: functii.h functii.c
        gcc -c functii.c

afis.o: afis.h afis.c
        gcc -c afis.c

myprog.o: myprog.c
        gcc -c myprog.c

.PHONY: clean
clean:
        rm hello myprog.o functii.o afis.o
```

La o compilare din mai multe fişiere, pot să apărea mai multe probleme prezentate în continuare.

10.7.2.1 Definiţii multiple pentru un simbol (funcţie sau variabilă)

Schema de mai jos descrie o situaţie ce se întâlneşte des şi cum poate fi ea evitată.



De obicei, într-un fişier header (`f_mare.h`) sunt incluse alte fişiere header (`f_mijlociu_1.h` şi `f_mijlociu_2.h`) în care sunt definite tipurile utilizate în definiţiile din fişierul iniţial (`f_mare.h`). Fişierele header (`f_mijlociu_1.h` şi `f_mijlociu_2.h`) utilizate de `f_mare.h` pot să includă la rândul lor header-e - în cazul defavorabil pot include acelaşi header (`f_mic.h`).

În mod normal, preprocesorul trebuie să producă un fişier `.c` complet pentru `fisier.c`. În condiţiile prezentate mai sus, el ajunge sa includă de două ori un fişier şi astfel apar definiţii multiple pentru simbolurile din fişierul inclus de mai multe ori.

Pentru a preveni multipla incluziune pentru fişierele header, este folosită următoarea construcţie:

```
$ cat header_file.h
#ifndef _HEADER_SOMETHING
#define _HEADER_SOMETHING

// conţinut efectiv

#endif
```

unde `_HEADER_SOMETHING` diferă între fişierele `.h`.

În cazul în care fişierul header nu a fost inclus niciodată, variabila `_HEADER_SOMETHING` nu este definită şi astfel se intră în corpul directivei `#ifndef`. În corpul directivei se defineşte `_HEADER_SOMETHING`. La încercarea următoare de a include acelaşi fişier, condiţia `#ifdef`-ului nu va mai fi îndeplinită, pentru că `_HEADER_SOMETHING` a fost definită anterior. În acest fel, definirea multiplă este prevenită.

10.7.2.2 Mai multe implementări pentru acelaşi simbol (funcţie sau variabilă)

Dacă un fişier header ar conţine implementarea unei funcţii şi două fişiere .c ar include acel fişier header, fişierele obiect rezultate din fişierele .c ar avea implementată acea funcţie. Dacă linker-ul utilizează ambele fişierele obiect pentru a crea un executabil, o să apară o eroare căci se vor găsi două implementări pentru acelaşi simbol (simbolul va fi definit de două ori).

Pentru a preveni această problemă, fişierele header nu conţin implementări, ci conţin doar referinţe externe la implementări. Acest lucru înseamnă că fişierul obiect rezultat dintr-o sursă .c care include header-ul, va avea marcate toate simbolurile ca fiind externe, mai puţin cele pe care compilatorul le-a descoperit definite în fişierul `.c`.

La link-editarea cu alte fişiere obiect, linker-ul va încerca să rezolve referinţele externe din fişierul obiect, căutând implementările în alte fişiere obiect.

Pentru a obţine acest comportament, simbolurile din fişierele header sunt prefixate de cuvântul cheie `extern`:

```
$ cat functii.h
//...
extern int mprod(int a, int b);
extern int mdiv(int a, int b);
//...
```

```
$ cat functii.c
#include "functii.h"

int mprod(int a, int b) { return a*b; }
int mdiv(int a, int b){ return ((b!=0)?a/b:a); }
```

Mai concret, se consideră secvenţa de compilare următoare:

```
$ gcc -c functii.c
```

În urma compilării va rezulta fişierul `funcţii.o` unde sunt **implementate** funcţiile definite în `functii.h` (`functii.c` implementează ceea ce este definit în `functii.h`).

```
$ gcc -c afis.c
```

În urma compilării va rezulta fişierul `afis.o` unde sunt implementate funcţiile definite în `afis.h`. (`afis.c` implementează ce este definit în `afis.h`).

```
$ nm afis.o
00000000 T afis
0000001b T endl
         U printf
         U putchar
```

Se observă:

- simboluri definite (T): funcţiile `afis`, `endl`
- simboluri nedefinite (U): `printf`, `putchar`

`putchar` se obţine deoarece compilatorul a optimizat codul `printf("\n")` şi a înţeles că se doreşte afişarea unui singur caracter „\n".

```
$ gcc -c myprog.c
```

În urma compilării va rezulta fişierul `myprog.o` unde sunt folosite doar referinţe către funcţiile (simbolurile) definite în `functii.h` şi `afis.h`.

```
$ nm myprog.o
         U afis
00000000 T main
         U mprod
```

Se observă:

- simboluri definite (T): funcţia main
- simboluri nedefinite (U): funcţiile apelate din celelalte fişiere obiect (afis, mprod)

```
$ gcc myprog.o functii.o afis.o -o hello
```

În etapa de compilare şi link-editare a executabilului final, link-editorul îşi dă seama că referinţele din `myprog.o` se leagă de implementările din `functii.o` şi `afis.o`.

```
$ nm hello | grep U
         U __libc_start_main@@GLIBC_2.0
         U printf@@GLIBC_2.0
         U putchar@@GLIBC_2.0
```

Se observă că au rămas simboluri nedefinite în executabilul final numit `hello`.

Motivul pentru care se obţine acest lucru este faptul că respectivele funcţii sunt externe aplicatiei şi se găsesc într-o biblioteca din sistem, respectiv biblioteca GNU Lib C (biblioteca C GNU) – simbolurile sunt marcate cu `@@GLIBC_2.0`.

10.7.3 Simularea executiei comenzii "make"

Pentru a realiza această simulare, se utilizează următorul fişier Makefile (comenzile sunt prefixate de caracterul `Tab`):

```
$ cat Makefile
pregatire: mancare curatenie
        touch pregatire
        @echo "Putem primi musafiri"
mancare: cumparaturi
        touch mancare
        @echo "Gata MANCARE"
curatenie:
        touch curatenie
        @echo "Gata CURATENIE"
cumparaturi:
        touch cumparaturi
        @echo "Gata CUMPARATURI"

.PHONY: pierde_cumparaturi
pierde_cumparaturi:
        rm cumparaturi
        @echo "pierdut CUMPARATURI"

.PHONY: clean
clean:
        rm cumparaturi
        rm curatenie
        rm mancare
        rm pregateste_musafiri
        @echo "Gata ANULARE"
```

Target-urile `pierde_cumparaturi` şi `clean` nu produc fişiere cu numele target-urilor şi au fost marcate cu `.PHONY`.

In acelaşi director în care a fost salvat fişierul `Makefile` anterior, se execută comanda:

```
$ make mancare
touch cumparaturi
Gata CUMPARATURI
touch mancare
Gata MANCARE
```

Observatii:

- este executat şi target-ul `cumparaturi` din lista de dependinţe pentru `mancare`
- o parte dintre target-uri conţin `touch numetarget` pentru ca `make` să considere că target-urile au fost făcute.
- target-urile `.PHONY` se execută de fiecare dată când sunt apelate
- `make` afişează comenzile din target-urile neprefixate cu `@`
- `make` afişează output-urile tuturor comenzilor

Dacă se încearcă execuţia target-ului cumparaturi se va observa că make detectează că acesta a fost executat (detecţia se face prin intermediul fişierului cumpărături care a fost creat la execuţia target-ului `cumparaturi`, prin comanda `touch cumparaturi`.

```
$ make cumparaturi
make: `cumparaturi' is up to date.
```

Se poate simula că s-au pierdut cumpărăturile prin:

```
$ make pierdut_cumparaturi
rm cumparaturi
Pierdut CUMPARATURI
```

Se pot face din nou cumpărăturile cu:

```
$ make cumparaturi
touch cumparaturi
Gata CUMPARATURI
```

Dacă încearcă execuția target-ului `mancare` se obține:

```
$ make mancare
touch mancare
Gata MANCARE
```

Target-ul `mancare` va fi rulat din nou, cu toate a fost executat cu succes înainte. Acest lucru s-a întâmplat pentru că `make` şi-a dat seama că o dependință (în acest caz `cumpărături`) s-a modificat.

Dacă se încearcă acum execuția target-ului pregătire se obține:

```
$ make pregatire
touch curatenie
Gata CURATENIE
touch pregatire
Putem primi musafiri
```

Se observă că doar task-urile nefăcute vor fi rulate.

De obicei, `Makefile`-urile au şi target-uri care anulează toate celălalte target-uri (în cazul compilării programelor, spre exemplu, se şterg fişierele obiect) – în cazul exemplului anterior, target-ul pentru anularea completă a modificărilor este `clean`.

Se poate testa întregul set de target-uri folosind:

```
$ make clean
rm cumparaturi
rm curatenie
rm mancare
rm pregatire
Gata ANULARE

$ make pregatire
touch cumparaturi
Gata CUMPARATURI
touch mancare
Gata MANCARE
touch curatenie
Gata CURATENIE
touch pregatire
Putem primi musafiri
```

Cuvinte cheie

- compilare
- executabil
- interpretare
- limbaj
- proces
- sursă
- preprocesare
- asamblare
- cod obiect
- linker
- bibliotecă
- partajare
- portabilitate
- execuţie
- apel de sistem
- make
- target
- dependinţă

Întrebări

1. Bibliotecile partajate de funcţii:

- ❑ sunt încărcate explicit la cerere de către aplicaţii
- ❑ ajută ca aplicaţiile să fie mai simplu de întreţinut
- ❑ măresc aplicaţiile deoarece sunt incluse în executabilele rezultate
- ❑ sunt prezente sub formă de arhive de fişiere obiect

2. Pentru a dezactiva link-editarea implicită pe care o realizează GCC-ul, se utilizează parametrul:

- ❑ `-O0`
- ❑ `-c`
- ❑ `-lstdc++`
- ❑ `-o`

3. Caracteristica unei aplicaţii de a putea fi folosită într-un mediu pentru care nu a fost iniţial proiectată:

- ❑ se întâlneşte doar la bibliotecile partajate
- ❑ este întâlnită numai la aplicaţiile Java
- ❑ se numeşte portabilitate
- ❑ nici una din variantele de mai sus

4. Pentru a executa un target dintr-un *Makefile*, se foloseşte comanda

- ❑ `make Makefile <target>`
- ❑ `make <target>`
- ❑ `<target> make`
- ❑ `Makefile <target>`

5. În urma preprocesării se obțin:
- fişiere obiect
- fişiere sursa C fără directive de compilare
- fişiere în limbaj de asamblare
- fişiere executabile

6. O bibliotecă de funcții neintegrată în executabil, care nu este încărcată la cererea explicită a programului, se numeşte:
- static library
- shared library
- dynamic library
- common library

7. Apelul de sistem reprezintă o metodă de a accesa resursele computerului. Apelurile de sistem sunt realizate doar de către biblioteci.
- adevărat, adevarat
- adevărat, fals
- fals, adevărat
- fals, fals

8. Fie comanda:

```
$ file /lib/libm-2.3.3.so
/lib/libm-2.3.3.so: ELF 32-bit LSB shared object, Intel 80386, version 1
(SYSV), not stripped
```

Care afirmație referitoare la fisierul `/lib/libm-2.3.3.so` este falsă:
- fişierul este binar, nu poate fi editat cu un editor text
- fişierul este o biblioteca partajata
- fişierul este executabil
- fişierul functioneaza pe o platforma Intel pe 32 biti

9. Programele interpretate se depanează mult mai greu. La compilare sunt produse fişiere în limbaj de asamblare care sunt executate direct pe procesor.
- adevărat, adevarat
- adevărat, fals
- fals, adevărat
- fals, fals

10. Apelurile de sistem ale programului `<program>` pot fi urmărite folosind comanda:
- `time <program>`
- `file <program>`
- `ldd <program>`
- `strace <program>`

11 Shell scripting

It is easier to port a shell than a shell script.
Larry Wall

Ce se învață din acest capitol?

- Rolul unui shell în sistem
- Ce este un script shell
- Utilitatea scripturilor shell
- Interacțiunea cu shell-ul; execuția comenzilor într-un shell
- Crearea şi rularea unui script shell
- Programarea shell: variabile, funcții, structuri de control
- Folosirea filtrelor de text
- Expandarea shell
- Programarea shell (batch programming) pe Windows
- Utilitarul sed

11.1 Noțiuni introductive

Capitolul de față îşi propune prezentarea mecanismelor prin care diversele comenzi utilizate până acum pot fi combinate pentru obținerea unor efecte complexe în cadrul unui script shell.

*Un **script shell** este un program scris într-un fişier text format din combinații de comenzi şi instrucțiuni specifice unui interepretor de comenzi (unui shell).*

Un shell este un program care asigură utilizatorului o interfață de control şi utilizare a sistemului de operare. Un shell poate avea interfață grafică (GUI) sau poate avea interfață text (CLI). În cea de-a doua situație un shell mai poartă denumire de interpretor de comenzi. Shell-uri cu interfață grafică sunt `Windows Explorer` sau medii desktop construite peste `X Window System`. Shell-uri cu interfață text (interpretoare de comenzi) sunt `cmd.exe` (Windows NT), `Windows PowerShell`, `Bash` (Bourne Again Shell), `csh` (C Shell).

Noțiunea de shell scripting se aplică shell-urilor cu interfață text (`Bash`, `C shell`, etc.). În capitolul de față, shell-ul folosit va fi Bash. GNU Bash (Bourne-Again Shell) este shell-ul implicit pe majoritatea distribuțiilor de Linux şi pe Mac OS X. Ne vom referi prin Bash (cu majusculă) la shell, iar cu `bash` la executabilul utilizat de acesta.

11.1.1 De ce shell scripting?

Utilitatea principală a unui shell script derivă din faptul că este o combinație de comenzi deja existente. Acest lucru înseamnă posibilitatea de automatizare. Sarcinile repetitive pot fi

descrise într-un script shell. Ori de câte ori este necesară executarea acelei sarcini, se rulează script-ul shell şi aceasta va fi îndeplinită.

Un exemplu elocvent este crearea unui cont de utilizator. Paşii urmaţi pentru crearea unui cont pot fi următorii:

- crearea unui cont de utilizator (folosind `useradd`);
- copierea unor directoare şi fişiere pentru contul proaspăt creat;
- stabilirea de cote (quota);
- schimbarea/stabilirea parolei;
- transmiterea unui e-mail pentru a notifica utilizatorul de crearea contului.

În lipsa unui script shell, administratorul sistemului ar trebui să ruleze comenzi pentru fiecare din paşii de mai sus ori de câte ori este necesară crearea unui cont de utilizator. Folosind un script shell, administratorul îl va rula doar pe acesta (folosind eventuali parametri). Efectul imediat este mărirea eficienţei.

Scripturi shell pot fi folosite pentru automatizarea sarcinilor zilnice: realizarea de backup-uri, verificarea stării sistemului, etc.

În altă situaţie, un utilizator doreşte realizarea unei noi comenzi prin îmbinarea unor comenzi deja existente. Pentru acest lucru, el va crea un script shell pe care-l va putea utiliza ulterior.

În sistemele Unix, scripturile shell au un grad mare de utilizabilitate pentru că cea mai mare partea a comenzilor urmează filozofia Unix: „*Do one thing, do one thing well*". Acest lucru însemnă că fiecare comandă este foarte utilă şi performantă la realizarea unei anumite sarcini, urmând ca sarcinile mai complexe să fie realizate prin „îmbinarea" mai multor comenzi simple într-un script shell.

11.1.2 Facilităţi oferite de scripturile shell

În afara folosirii comenzilor simple deja existente în sistem, lucru care conduce la posibilitatea de automatizare, un script shell oferă o serie de alte facilităţi utilizatorului.

În primul rând, un shell oferă un limbaj de programare cu variabile şi instrucţiuni de ciclu (`for`, `while`) şi de decizie (`if`, `case`). Acest lucru înseamnă că, dincolo de comenzile de bază, un script shell poate folosi facilităţile tipice unui limbaj de programare.

Evident, se poate pune întrebarea: De ce nu s-ar folosi un limbaj de programare precum C/Java?

Avantajul major unui script shell este timpul redus de scriere a acestuia, utilizarea unor componente deja create, fără a fi nevoie de rescrierea acestora şi folosirea unor resurse specifice ale sistemului: fişiere de configurare, drepturi de acces, utilizatori, etc.. Folosind comenzi deja existente, facilităţi de combinare a acestora şi de control al fluxului, un script shell este o soluţie rapidă şi eficientă a unei probleme. Se spune că un script shell este o metodă „*quick and dirty*" de rezolvare a unor probleme care ar necesita efort mai mare dacă s-ar folosi alte limbaje. În plus depanarea unui script shell este extrem de facilă. În momentul în care un script shell nu funcţionează corect se poate determina şi corecta foarte rapid linia din script care a cauzat eroarea.

Dezavantajul unui script shell este viteza scăzută de execuţie şi lipsa accesului la structuri eficiente de calcul. Astfel, nu este recomandată folosirea unui script shell într-o situaţie în care viteza de execuţie este un factor important. La fel, anumite probleme nu se pretează la a fi rezolvate cu un script shell din cauza lipsei de facilităţi. Problemele de algoritmică, efectuarea de calcule complexe, interacţiunea cu hardware-ul se realizează prin folosirea de limbaje specializate.

Totodată, un script shell permite interacţiunea între comenzi (sau mai bine zis între procesele create prin rularea unei comenzi), prin operatori specifici: `|` (pipe), `;`, `||`, `&&`, etc.

Alte facilităţi includ:

- posibilitatea reținerii ieşirii unei comenzi într-o variabilă pentru folosirea ulterioară a acesteia;
- posibilitatea redirectării intrării/ieşirii unei comenzi dintr-un/într-un fişier; acest lucru înseamnă prelucrarea foarte uşoară a fişierelor cu relevanță în sistem (`/etc/passwd`, `/etc/shadow`, etc.);
- folosirea de comenzi non-interactive şi programarea scriptului pentru execuție la un moment ulterior.

Se observă, aşadar, că un script shell este un instrument foarte puternic pentru rezolvarea rapidă a diverselor probleme des întâlnite de utilizator. Programarea shell devine un factor important în mai buna utilizare şi gestiune a sistemului.

11.2 Interacțiunea cu shell-ul

Shell-urile cu interfață text interacționează cu utilizatorul prin intermediul comenzilor. Un shell oferă un prompt utilizatorului (în mod tipic acesta se termină cu un caracter de forma `$`, `%` sau `#`) unde se pot introduce comenzi. Execuția unei comenzi înseamnă, de obicei, generarea unui proces dintr-un executabil asociat comenzii.

O deprindere utilă în lucrul cu comenzile este prelucrarea şirului de caractere care le formează. Astfel, un utilizator avansat va edita comenzile sale şi va folosi istoricul pus la dispoziție de un shell pentru a fi cât mai eficient. Shell-ul pune la dispoziția utilizatorului mai multe facilități de editare şi utilizare rapidă a comenzilor.

11.2.1 Editarea comenzilor

O comandă este introdusă la promptul shell-ului prin folosirea tastelor corespunzătoare, după care se apasă `ENTER` şi comanda va fi executată.

Există, însă, situații în care se greşeşte sau se uită un argument sau o literă şi este nevoie de editarea acelei comenzi pentru obținerea efectului dorit. Editarea comenzii poate însemna adăugarea sau şteregerea unui caracter, a unui set de caractere, a unui argument al comenzii. În mod evident, la fel ca în cazul unui editor, este nevoie şi de parcurgerea comenzii scrise.

Editarea comenzilor în Bash este realizată prin intermediul bibliotecii **readline**. Detalii despre aceasta găsiți la

http://www.gnu.org/software/bash/manual/bashref_81.html#SEC88.

Biblioteca readline este cea care asigură editarea unei comenzi, oferind interfața de control a terminalului.

Comenzile puse la dispoziție de `readline` pentru editarea comenzilor shell sunt inspirate din editorul **Emacs**. Urmând convenția Emacs, vom prescurta cu `C` tasta `Control` iar cu `M` tasta `Alt` (`Meta`). Prezentăm, în continuare, câteva dintre cele mai utile comenzi:

- `C-f`, `C-b` – un caracter înainte, respectiv înapoi (echivalent cu săgeată dreapta şi săgeată stânga)
- `C-a`, `C-e` – început de linie, respectiv sfârşit de linie (echivalent cu `Home` şi `End`)
- `M-f`, `M-b` – cuvânt înainte, respectiv înapoi; pe unele terminale (spre exemplu `gnome-terminal`) trebuie dezactivate scurtăturile terminalului pentru a permite comenzile readline
- `C-d` – şterge caracterul de sub cursor (echivalent cu `Delete`)
- `Backspace` – şterge caracterul de dinainte de cursor
- `M-d` – şterge până la sfârşitul cuvântului
- `M-Backspace` – şterge până la începutul cuvântului
- `C-k` – şterge până la sfârşitul liniei

- `C-w` – şterge până la începutul liniei
- `C-y` – lipeşte (paste, yank) ceea ce s-a şters ultima oară
- `C-_` (Control – underscore) - anulează (undo) ultima comandă de editare.

Pentru cei ce doresc acest lucru, editarea comenzilor folosind readline poate fi personalizată folosind un fişier de configurare. Pentru mai multe detalii consultaţi manualul (http://www.gnu.org/software/bash/manual/bashref.html#SEC97).

11.2.2 Folosirea istoricului. Completare automată

Biblioteca readline memorează comenzile folosite în shell într-un istoric care poate fi folosit ulterior. Se pot folosi comenzi readline pentru parcurgerea comenzilor shell. Astfel `C-p` sau `C-n` parcurg comenzile anterioare sau ulterioare comenzii curente. Sunt echivalente cu săgeată sus şi săgeată jos.

Istoricul este salvat la părăsirea shell-ului într-un fişier. În cazul bash acesta este `~/.bash_history`. La o nouă autentificare, shell-ul va încărca respectivele comenzi din history şi utilizatorul le va putea reutiliza.

11.2.2.1 Căutarea comenzilor (reverse search)

În afara parcurgerii comenzilor din istoric, o opţiune foarte utilă este căutarea acestora. Pentru căutarea comenzilor în istoric se poate folosi comanda `C-r` (reverse search):

```
razvan@anaconda:~$ cd
(reverse-i-search)`chm': chmod o+w uploads/
```

În exemplul de mai sus, apăsarea `C-r` duce la schimbarea promptului. Utilizatorul poate introduce un şir de căutare pentru identificarea unei comenzi. Pentru fiecare caracter introdus, shell-ul va afişa prima comandă anterioră cea mai potrivită. Completarea şirului va duce la refacerea căutării. În afara introducerii şirului de căutare există câteva comenzi utile în modul de căutare:

- dacă se apasă (din nou) `C-r` se caută următoarea potrivire;
- dacă se apasă `ESC` sau `C-j` se încheie căutarea; comanda afişată la prompt va fi cea găsită;
- dacă se apasă `ENTER` se încheie căutarea şi se execută comandă găsită;
- dacă se apasă `C-g` se întrerupe căutarea cu revenire la comanda de dinainte de căutare.

11.2.2.2 Completare automată (autocompletion)

O facilitate extrem de utilă în eficientizarea lucrului cu comenzile shell este cea de completare automată (autocompletion). Această facilitate înseamnă introducerea unui şir parţial de început al unei comenzi sau al unei intrări în sistemul de fişiere şi apoi apăsarea tastei `TAB` pentru completarea automată. Spre exemplu, dacă cineva doreşte să verifice server-ul de nume (DNS) al unui sistem, va folosi comanda:

```
$ cat /etc/resolv.conf
```

Un calcul simplu arată că este nevoie de apăsarea a 21 de taste (cu `ENTER`) pentru execuţia comenzii.

Putem folosi, în schimb, completarea automată şi vom apăsa tastele în mod prezentat mai jos:

```
$ cat /e<TAB>res<TAB>.<TAB><ENTER>
```

Un calcul la fel de simplu indică apăsarea a doar 14 taste (plus că tasta TAB devine foarte importantă şi este apăsată mult mai uşor). Comanda afişată la prompt va fi identică cu cea de mai sus şi, drept urmare, efectul va fi acelaşi.

În consecinţă, se recomandă insistent folosirea facilităţii de completarea automată. Această facilitate nu este prezentă numai la shell-urile unui sistem de operare ci şi pentru comanda dispozitivelor de reţea şi a altor echipamente care dispun de o interfaţă în linia de comandă. În plus, folosirea tastei TAB are un rol important în evitarea erorilor de scriere în cazul intrărilor sistemului de fişiere. Experienţa personală a autorilor acestei cărţi a surprins, nu de puţine ori, crearea de fişiere cu nume „apropiate", dar nu identice, de cele reale (precum /etc/rezolv.conf, /etc/resolv.comf, /etc/resolv.cf, etc.). Aceste erori ar fi putut fi evitate (dincolo de o mai mare viteză de utilizare a comenzilor) prin folosirea facilităţii de completare automată.

În situaţia în care există mai multe posibilităţi de completare, apăsarea tastei TAB nu are nici un efect. Totuşi, se pot vizualiza toate aceste combinaţii prin apăsarea de două ori a tastei TAB. Astfel, apăsarea z<TAB><TAB> va duce la afişarea tuturor comenzilor care încep cu litera z:

```
razvan@anaconda:~$ z<TAB><TAB>
zcat          zegrep        zforce        zipgrep       zless
zcmp          zeisstopnm    zgrep         zipinfo       zmore
zdiff         zenity        zip           zipnote       znew
zdump         zfgrep        zipcloak      zipsplit      zsoelim
```

La fel, folosirea comenzii cat /<TAB><TAB> duce la afişarea tuturor intrărilor care se găsesc în directorul rădăcină al sistemului de fişiere:

```
razvan@anaconda:~$ cat /
.viminfo        home/           mnt/            tmp/
aquota.user     initrd/         opt/            usr/
bin/            initrd.img      proc/           var/
boot/           initrd.img.old  root/           vmlinuz
cdrom/          lib/            sbin/           vmlinuz.old
dev/            lost+found/     srv/
etc/            media/          sys/
```

Extinderea completării automate

După cum s-a observat, completarea automată se referă la comenzi şi la intrări în sistemul de fişiere. Totuşi, aceasta poate fi extinsă prin intermediul unor scripturi personalizate la argumentele posibile ale unei comenzi.

Astfel fişierul /etc/bash_completion este folosit pentru extinderea completării automate. Pentru folosirea acestui script este necesară rularea comenzii:

```
$ . /etc/bash_completion
```

Câteva linii din acest fişier:

```
complete -f -X '!*.@(sxi|sti|pps|ppt|pot|odp|otp)' ooimpress
```

Linia de mai sus înseamnă că dacă un utilizator foloseşte în linia de comandă ooimpress, argumentele posibile ale acesteia vor fi considerate doar fişierele cu extensiile precizate.

```
# user commands see only users
complete -u su usermod userdel passwd chage write chfn groups slay w
```

Linia de mai sus permite ca argumente numai nume de utilizatori pentru comenzile de lucru cu aceştia.

În afara extinderilor existente în /etc/bash_completion, utilizatorul îşi poate defini noi extinderi pentru a le integra în shell. Se recomandă ca acestea să fie plasate în directorul

`/etc/bash_completion.d`. Generarea de noi extensii de completare depăşeşte domeniul de interes al acestei cărţi. Recomandăm celor interesaţi pagina web asociată proiectului http://www.caliban.org/bash/index.shtml sau articolul http://www.debian-administration.org/articles/316.

11.2.3 Comenzi interne (built-in) şi comenzi externe

Comenzile care pot fi rulate dintr-un shell sunt de două tipuri:

- **comenzi externe**: sunt rulate prin intermediul unui executabil
- **comenzi interne**: sunt executate direct de shell fără invocarea unui alt program.

Comenzile interne (sau built-in) sunt utilizate pentru implementarea de facilităţi care sunt fie imposibil fie neconvenabil de realizat folosind programe separate. Un bun exemplu este comanda `cd` de schimbare a directorului curent. Această comandă nu poate fi folosită dintr-un executabil extern, întrucât la sfârşitul rulării comenzii s-ar pierde orice efect ar avea (şi nu ar fi posibilă, astfel, schimbarea persistentă a directorului curent).

Exemple de comenzi externe şi executabilul folosit pentru rularea acestora sunt:

- `ls` cu `/bin/ls`
- `file` cu `/usr/bin/file`
- `su` cu `/bin/su`
- `ifconfig` cu `/sbin/ifconfig`

Exemple de comenzi interne sunt:

- `:` - comanda nu face nimic, doar expandează variabile şi realizează redirectările; astfel, rularea comenzii:

```
$ : > out.txt
```

duce la trunchierea fişierului `out.txt`. Versiunea şi mai simplă este

```
$ > out.txt
```

- `.` – comanda este folosită pentru interpretarea unui fişier (de obicei script) în shell-ul curent. Echivalentă este comanda `source`. Un exemplu de folosire este încărcarea configuraţiilor din `/etc/bash_completion`:

```
$ . /etc/bash_completion
```

sau

```
$ source /etc/bash_completion
```

- `cd` – comanda folosită pentru schimbarea directorului curent
- `echo` – afişarea şirului de caractere primit ca parametru la ieşirea standard
- `alias/unalias` – configurarea/dezactivarea configurării unui alias pentru co comandă. Astfel, dacă folosim:

```
alias mygrep='grep --color=tty -d skip'
```

- comanda `mygrep` (alias) va fi lansată ca o comandă `grep` cu argumentele precizate; se poate folosi ca alias chiar numele comenzii, în cazul în care utilizatorul doreşte rularea implicită cu acele argumente:

```
alias grep='grep --color=tty -d skip'
```

De notat este faptul că o comandă poate fi implementată atât în shell (built-in) cât şi prin intermediul unui executabil. Astfel de comenzi sunt `echo` şi `time`. În aceste situaţii comanda executată implicit este comanda internă. Pentru execuţia comenzii externe trebuie apelată calea

completă. În exemplul de mai jos sunt prezentate cele două tipuri de rulări pentru comanda `time` (internă şi externă):

```
razvan@anaconda:~/code$ time ls
asm  boot_hello  latex  stack_ovfl  static  tests

real    0m0.004s
user    0m0.000s
sys     0m0.000s
razvan@anaconda:~/code$ /usr/bin/time ls
asm  boot_hello  latex  stack_ovfl  static  tests
0.00user 0.00system 0:00.00elapsed 133%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+254minor)pagefaults 0swaps
```

Se observă că poate exista o deosebire între formatele de ieşire ale unei comenzi interne, respectiv externe.

De asemenea, mecanismele de ajutor folosite pentru comenzi (`whereis`, `apropos`, `--help`, `man`, `info`) funcţionează doar pentru comenzi externe. Pentru comenzi interne va trebui consultată documentaţia de Bash (`man bash`, `info bash`) sau folosită comanda (internă) `help`:

```
razvan@anaconda:~/code$ help time
time: time [-p] PIPELINE
    Execute PIPELINE and print a summary of the real time, user CPU time,
    [...]
razvan@anaconda:~/code$ help alias
alias: alias [-p] [name[=value] ... ]
    `alias' with no arguments or with the -p option prints the list
    [...]
```

***NOTĂ**: O listă completă a comenzilor interne Bash găsiţi în pagina de manual. (man/info bash sau http://www.gnu.org/software/bash/manual/bashref.html#SEC55)*

11.2.4 Execuţia unei comenzi shell

După cum s-a precizat, comenzile sunt de două tipuri: interne şi externe. Clasificarea este dată de modul în care comenzile se execută în cadrul shell-ului. Astfel, **comenzile interne** (built-in) sunt executate direct în cadrul shell-ului, altfel spus în cadrul procesului curent. De partea cealaltă, execuţia unei **comenzi externe** înseamnă crearea unui proces fiu în shell-ul curent şi înlocuirea imaginii procesului cu executabilul asociat comenzii. Mai multe detalii despre caracteristicile şi crearea unui proces găsiţi în capitolul 4.

Un proces astfel creat va conţine ca imagine executabilul asociat comenzii. De obicei, shell-ul va aştepta terminarea procesului curent după care va oferi promptul utilizatorului. Acest efect poate fi schimbat prin rularea procesului în fundal (background) cu ajutorul operatorului `&` [vezi 4.3.1].

De reţinut este faptul că, în cazul înlănţuirii mai multor comenzi (vezi 11.3.3.2), se creează un proces pentru fiecare comandă. Astfel, la rularea comenzii:

```
$ ls -l | sort
```

se vor crea două procese: unul generat din executabilul `/bin/ls` iar altul generat din executabilul `/usr/bin/sort`. Shell-ul este procesul părinte al celor două procese şi va asigura comunicaţia între acestea (ieşirea comenzii `ls -l` va fi redirectată către intrarea comenzii sort).

Imaginea de mai jos ilustrează grafic execuţia comenzii de mai sus:

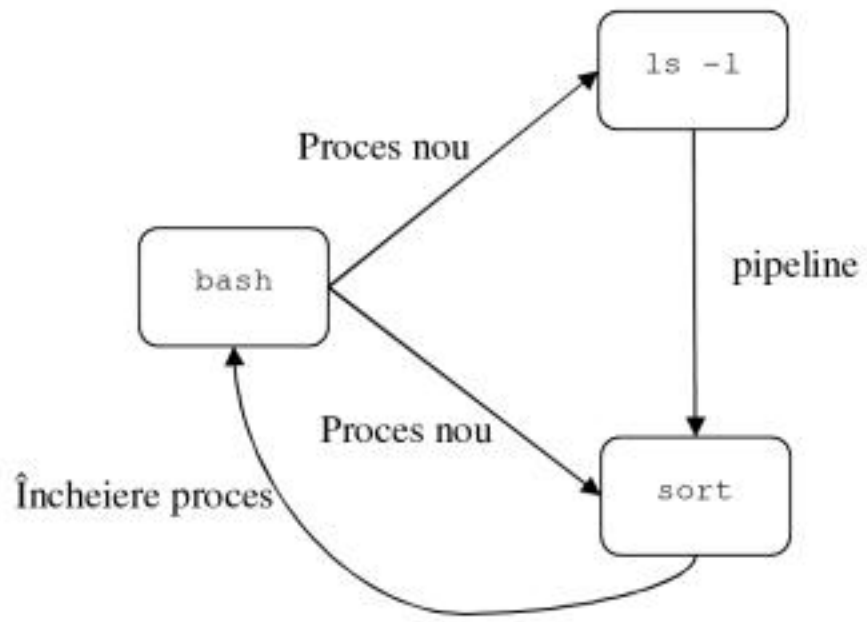


Figura11-1 - Execuția unei comenzi în shell

11.3 Scripturi shell

După cum am specificat, un script shell este un fişier text (un program) conținând comenzi shell, variabile, structuri de control şi funcții cu rolul de rezolvare a unei probleme. Un script shell este interpretat de shell. Interpretarea înseamnă parcurgerea fiecărei înlănțuiri de comenzi sau structuri de control şi realizarea sarcinilor descrise de aceasta.

Ceea ce separă un script shell de limbajele interpretate obişnuite este folosirea directă a comenzilor shell, a facilităților de înlănțuire a comenzilor puse la dispoziție şi a fişierelor de configurare specifice sistemului.

Comenzile componente ale unui script shell pot fi executate manual la promptul shell-ului cu aceleaşi rezultate ca în cazul rulării scriptului. Astfel, liniile următoare dintr-un script shell:

```
if test 10 -gt 4; then
    echo „mai mare”
else
    echo „mai mic”
fi
```

pot fi executate direct în linia de comanda:

```
razvan@anaconda:~$ if test 10 -gt 4; then echo "mai mare"; else echo "mai
mic"; fi
mai mare
```

sau astfel:

```
razvan@anaconda:~$ if test 10 -gt 4
> then
> echo "mai mare"
> else
> echo "mai mic"
> fi
mai mare
```

În ultimul exemplu, shell-ul oferă utilizatorului un miniprompt (>) pentru a indica acestuia că nu a finalizat comanda. În cazul de față comanda este if. Detalii despre if şi despre efectul acestei bucăți de script se regăsesc în secțiunea 11.4.3.1.

11.3.1 Cel mai simplu script shell

Pentru a păstra tonul tutorialelor de limbaje de programare, cel mai simplu script shell pe care îl vom descrie va fi cel care va afişa mesajul „Hello, World!" la ieşirea standard. Programul este prezentat mai jos:

```
echo "Hello, World!"
exit 0
```

Programul este foarte simplu. Folosim comanda echo pentru a afişa şirul de caractere primit ca argument la ieşirea standard şi comanda exit pentru ca programul să se întoarcă cu succes.

Folosirea comenzii exit este opțională; un script shell va întoarce în mod implicit `0` dacă se ajunge la sfârşitul programului. Astfel, scriptul nostru shell poate fi scris într-o singură linie:

```
echo "Hello, World!"
```

11.3.1.1 Rularea unui script shell

Rularea unui script shell se realizează la fel ca în cazul unui limbaj interpretat prin transmiterea sa ca parametru interpretorului:

```
razvan@anaconda:~/junk/scripting$ bash hw.bash
Hello, World!
```

Scriptul shell poartă denumirea de hw.bash. Se recomandă ca scripturile shell Bash să aibă extensia .sh sau .bash, cu toate că în lumea Unix extensia poate fi omisă. Se observă afişarea mesajului „Hello, World!" la ieşirea standard.

În spatele acestei execuții, shell-ul generează un nou proces care interpretează conținutul scriptului primit ca parametru.

11.3.1.2 #! – shebang

Mecanismul uzual de execuție a unui shell script este acela prin care se execută un program obişnuit:

```
$ ./hw.bash
```

Totuşi, pentru a putea realiza acest lucru avem de îndeplinit două precondiții:

- scriptul trebuie să fie executabil

în momentul execuției trebuie să se cunoască ce interpretor va fi folosit (`/bin/bash` pentru Bash, `/usr/bin/python` pentru Python, `/usr/bin/perl` pentru Perl, etc.)

Primul pas se realizează foarte simplu. Se adaugă drepturi de execuție folosind comanda `chmod`:

```
razvan@anaconda:~/junk/scripting$ chmod a+x hw.bash
razvan@anaconda:~/junk/scripting$ ls -l hw.bash
-rwxr-xr-x 1 razvan razvan 42 Aug 31 18:46 hw.bash
```

Pentru realizarea celui de-al doilea pas se foloseşte perechea de caractere „**#!**" (diez şi semnul exclamării) denumită şi **shebang**. Această pereche de caractere este urmată de calea completă către programul care va interpreta scriptul. Linia care conține shebang şi calea completă către interpretor este prima linie din script. În momentul în care scriptul este lansat în execuție, shell-ul parcurge prima linie şi foloseşte interpretorul specificat pentru a executa scriptul. Exemple de linii shebang sunt:

- `#!/bin/bash` – executarea scriptului folosind Bash
- `#!/usr/bin/perl` – executarea scriptului folosind Perl
- `#!/usr/bin/php` – executarea scriptului folosind PHP
- `#!/usr/bin/python` – executarea scriptului folosind Python

Astfel, după adăugarea drepturilor de execuție şi a liniei shebang scriptului nostru:

```
razvan@anaconda:~/junk/scripting$ cat hw.bash
#!/bin/bash

echo "Hello, World!"   # afisare mesaj
exit 0
```

îl putem executa:

```
razvan@anaconda:~/junk/scripting$ ./hw.bash
Hello, World!
```

NOTĂ: Un script shell nu necesită prezența liniei ***shebang****. În mod implicit un script este interpretat folosind /bin/bash. Se recomandă, totuşi, prezența acestei linii pentru a asigura consistența.*

11.3.2 Comentarii într-un script shell

Comentariile într-un script shell se realizează cu ajutorul caracterului #. Un comentariu începe de la apariția # până la sfârşitul liniei, asemănător cu // din C++. Exemple de comentarii sunt:

```
# comentariu
echo "hello"  # comentariu
# echo "hello"  (comanda comentata)
```

11.3.3 Comenzi simple shell

În mod evident, scriptul shell prezentat anterior nu este un script shell în adevăratul sens al cuvântului. Aceasta deoarece acelaşi efect poate fi obținut prin rularea simplă a comenzii echo urmată de argumentul „`Hello, World!`" la promtpul shell-ului. Un script shell va conține mai multe comenzi care vor fi interpretate într-o singură instanță în momentul execuției scriptului.

Un script shell va folosi de multe ori un set comenzi interne. Vom prezenta, în continuare, o parte din cele mai folositoare comenzi interne:

11.3.4 echo

Comanda echo permite afişarea şirului primit ca parametru la ieşirea standard a terminalului:

```
razvan@anaconda:~$ echo "test"
test
```

Opțiuni utile pentru echo sunt:

- `-n`: dezactivează afişarea implicită a unui caracter newline la sfârşitul liniei:

```
razvan@anaconda:~$ echo -n "test"
testrazvan@anaconda:~$
```

- `-e`: permite interpretarea caracterelor speciale:

```
razvan@anaconda:~$ echo -ne "test\n\t\ttest2\n"
test
                test2
```

11.3.4.1 printf

Comanda `printf` permite, ca şi `echo`, afişarea unui şir de caractere la ieşirea standard. Permite, însă, la fel ca funcția `printf` din C, formatarea şirului de afişare:

```
razvan@anaconda:~$ printf "%s    %03d\n" "test" 20
test    020
```

Se observă similaritatea cu funcția `printf` din C: primul argument este șirul de formatare iar celelalte sunt argumentele de afișat la ieșirea standard.

11.3.4.2 exit

Comanda `exit` permite întreruperea execuției scriptului. Comanda poate primi un argument care specifică valoarea de retur. Această valoare este implicit `0`.

11.3.4.3 export

Comanda export permite exportarea unei variabile (sub forma unei variabile de mediu) din scriptul curent în shell. Astfel, după ce shell-ul a încheiat interpretarea scriptului, variabilă va putea fi folosită direct din shell. Pentru mai multe detalii despre variabile consultați secțiunea 11.4.1.

11.3.4.4 set

Comanda `set` este utilizată pentru configurarea și personalizarea shell-ului. O opțiune utilă este opțiunea `-x` care permite afișarea comenzilor unui script shell pe măsură ce acestea sunt executate.

11.3.4.5 read

Comanda `read` este folosită pentru a citi informații de la intrarea standard și stocarea acesteia într-o variabilă. În exemplul de mai jos, se dorește stocarea informației introduse de utilizator în variabila `a`:

```
razvan@anaconda:~/junk/scripting$ read a
test_msg
razvan@anaconda:~/junk/scripting$ echo $a
test_msg
```

Se observă că variabila conține șirul `test_msg` introdus de utilizator.

11.3.5 One liners

Multe din problemele ce necesită utilizarea facilităților shell pot fi realizate prin intermediul unei simple linii. S-a observat că nu este necesar (și nici eficient) să utilizăm un script shell pentru afișarea unui mesaj `„Hello, World!\n"` la ieșirea standard.

Folosind redirectări sau înlănțuirea comenzilor, se pot obține efecte imediate într-o singură linie de shell. Astfel de comenzi combinate poartă numele de one liners.

11.3.5.1 Redirectarea intrării/ieșirii unei comenzi

Redirecționarea unei comenzi se oferă la posibilitatea de substituire a intrării, ieșirii sau erorii standard a unei comenzi cu un fișier. Astfel, redirectarea intrării standard înseamnă că o comandă va citi informații dintr-un fișier în locul citirii de la intrarea standard. Redirectarea ieșirii standard înseamnă că rezultatul comenzii se va regăsi într-un fișier de ieșire.

Pentru redirectarea intrării standard se folosește operatorul `<` urmat de numele fișierul din care se realizează redirectarea. În cazul redirectării ieșirii standard operatorul folosit este `>`. Informații despre redirectarea intrării și ieșirii standard se regăsesc și în capitolul 3.

Vom exemplifica redirectarea folosind utilitarul `cat`. Comanda `cat` realizează, în mod implicit, afișarea conținutului intrării standard la ieșirea standard. Un exemplu de rulare este prezentat în continuare (liniile scrise cu font cursiv reprezintă ieșirea):

```
razvan@anaconda:~$ cat
mesaj1
mesaj1
mesaj2
mesaj2
```

Pentru încheierea introducerii de informații de la intrarea standard trebuie folosită combinația `CTRL-D`, însemnând sfârșit de fișier (end of file). Se observă că ceea ce introduce utilizatorul la intrarea standard se afișează la ieșirea standard.

Dacă vom redirecta un fișier la intrarea standard, efectul va fi afișarea conținutului fișierului la ieșirea standard:

```
razvan@anaconda:~/junk/scripting$ cat < hw.bash
#!/bin/bash

echo "Hello, World!"

exit 0
```

Acest lucru este echivalent cu transmiterea fișierului ca argument comenzii `cat`.

În mod similar, dacă redirectăm ieșirea standard într-un fișier, acesta va conține ceea ce a introdus utilizatorul la intrarea standard (se folosește `CTRL-D` pentru încheiere):

```
razvan@anaconda:~/junk/scripting$ cat > out.txt
introducem mesaj
dupa care apasam CTRL-D
razvan@anaconda:~/junk/scripting$ cat out.txt
introducem mesaj
dupa care apasam CTRL-D
```

În exemplul de mai sus, comanda cat a afișat fișierul out.txt prin transmiterea acestuia ca argument, echivalent cu redirectarea acestuia către intrarea standard a comenzii. Acest comportament se extinde și la alte comenzii de prelucrare de fișiere text (vezi 11.5).

Un efect care poate părea surprinzător este folosirea simultană a redirectării intrării și ieșirii standard. În acest fel, conținutul fișierului de la intrare este copiat în fișierul de la ieșire. Efectul este identic cu folosirea comenzii `cp`:

```
razvan@anaconda:~/junk/scripting$ cat < hw.bash > new.bash
razvan@anaconda:~/junk/scripting$ cat new.bash
#!/bin/bash

echo "Hello, World!"

exit 0
```

Pentru redirectarea erorii standard se folosește operatorul `2>`. Aceasta are legătură cu descriptorul de fișier asociat: intrarea standard are asociat descriptorul de fișier `0`, ieșirea standard are asociat descriptorul de fișier `1` iar eroarea standard are asociat descriptorul de fișier `2`.

Astfel, dacă se doresc redirectate într-un fișier erorile și avertismentele emise de `gcc` se folosește operatorul `2>`:

```
razvan@anaconda:~/junk/scripting$ gcc hw.c 2> err_warn.txt
razvan@anaconda:~/junk/scripting$ cat err_warn.txt
hw.c: In function 'main':
hw.c:3: warning: incompatible implicit declaration of built-in function 'printf'
razvan@anaconda:~/junk/scripting$ gcc hw.c 2> /dev/null
```

În cel de-al doilea exemplu, s-a redirectat eroarea standard către `/dev/null`. Aceasta înseamnă că nu se dorește afișarea la eroarea standard sau într-un fișier a diverselor informații de avertisment afișate de `gcc`.

Se poate redirecta atât ieşirea standard cât şi eroarea standard folosind operatorul `&>`. În exemplu de mai jos se redirectează atât ieşirea standard cât şi eroarea standard a unei rulări a comenzii `strace`:

```
razvan@anaconda:~/junk/scripting$ strace ls &> out_ls.txt
```

Operatorul >>

Operatorul `>>` este utilizat pentru redirectarea ieşirii standard însă cu adăugarea (append) a informațiilor redirectate la sfârşitul fişierului. Echivalent există operatorul `2>>` pentru adăugarea informațiilor de la eroarea standard. În exemplul de mai jos scriem ieşirile comenzilor `ls`, `ps` şi `uptime` într-un acelaşi fişier:

```
razvan@anaconda:~/junk/scripting$ ls > out.txt
razvan@anaconda:~/junk/scripting$ ps >> out.txt
razvan@anaconda:~/junk/scripting$ uptime >> out.txt
razvan@anaconda:~/junk/scripting$ cat out.txt
a.out
err_warn.txt
hw.bash
hw.c
new
new.bash
out.txt
out_ls.txt
  PID TTY          TIME CMD
30624 pts/2    00:00:00 bash
31508 pts/2    00:00:00 ps
 15:34:59 up 59 days, 39 min,  2 users,  load average: 0.37, 0.55, 0.61
```

Here documents; operatorul <<

Operatorul `<<` este folosit în ceea ce se cheamă **here document**. Un „here document" este folosit, de obicei, cu utilitare interactive. Formatul de utilizare este:

```
command <<word
```

Acest lucru va impune comenzii citirea informației aflată în continuarea comenzii de la intrarea standard până la întâlnirea cuvântului `word` în corpul scriptului. În exemplul de mai jos, se citeşte de la intrarea standard pana la întâlnirea cuvântului `END`:

```
razvan@anaconda:~/junk/scripting$ cat here.bash
#!/bin/bash

cat <<END
mesaj simplu
... pentru un here document ...
avantajul este ca nu trebuie sa folosesc multe comenzi echo
END

echo "s-a terminat"
razvan@anaconda:~/junk/scripting$ bash here.bash
mesaj simplu
... pentru un here document ...
avantajul este ca nu trebuie sa folosesc multe comenzi echo
s-a terminat
```

După cum se observă şi din comentariile fişierului, un „here document" poate fi folosit în locul comenzii `echo`. Un exemplu util îl constituie generarea unei pagini web simple. O primă alternativă este folosirea comenzii `echo`:

```
razvan@ragnarok:~/junk/scripting$ cat echo.bash
#!/bin/bash

hostname="ragnarok"
name="razvan"

echo -e "<html>"
echo -e "\t<title>"
echo -e "\t\tPagina mea"
echo -e "\t</title>"
echo -e "\t<body>"
echo -e "\t\t<h3>Statia este $hostname</h3>"
echo -e "\t\t<h3>Numele meu este $name</h3>"
echo -e "\t\t<p>Data este $(date)</p>"
echo -e "\t</body>"
echo -e "</html>"
razvan@ragnarok:~/junk/scripting$ bash echo.bash
<html>
        <title>
                Pagina mea
        </title>
        <body>
                <h3>Statia este ragnarok</h3>
                <h3>Numele meu este razvan</h3>
                <p>Data este Tue Sep  4 23:23:05 EEST 2007</p>
        </body>
</html>
```

Se observă că este incomodă folosirea comenzii `echo`. O soluție este utilizarea unui „here document":

```
razvan@ragnarok:~/junk/scripting$ cat here.bash
#!/bin/bash

hostname="ragnarok"
name="razvan"

cat <<EndOfHtml
<html>
        <title>
                Pagina mea
        </title>
        <body>
                <h3>Statia este $hostname</h3>
                <h3>Numele meu este $name</h3>
                <p>Data este $(date)</p>
        </body>
</html>
EndOfHtml
```

Informații suplimentare despre „here documents" găsiți la adresa http://tldp.org/LDP/abs/html/here-docs.html.

Here strings; operatorul <<<

Operatorul `<<<` permite folosirea de here strings, adică redirectarea unui şir de caractere către intrarea standard a unei comenzi. În exemplul de mai jos, se foloseşte comanda read pentru a citi conținutul unui şir într-un set de variabile:

```
razvan@anaconda:~$ read a b c d <<< "alfa beta gamma delta "
razvan@anaconda:~$ echo $a $b $c $d
alfa beta gamma delta
```

Ca şi în cazul operatorului <<, acelaşi rezultat poate fi obținut prin folosirea comenzii `echo`:

```
razvan@anaconda:~$ echo "alfa beta gamma delta" | read a b c d
```

Trunchierea unui fişier

Trunchierea unui fişier înseamnă eliminarea conținutului acestuia: fişierul devine gol (dimensiune `0`). Trunchierea unui fişier se realizează prin redirectarea conținutului `/dev/null` în fişier. Acest lucru se realizează folosind comanda:

```
razvan@anaconda:~/junk/scripting$ ls -l out.txt
-rw-r--r-- 1 razvan razvan 220 Sep  4 15:34 out.txt
razvan@anaconda:~/junk/scripting$ cat /dev/null > out.txt
razvan@anaconda:~/junk/scripting$ ls -l out.txt
-rw-r--r-- 1 razvan razvan 0 Sep  4 15:35 out.txt
```

Mai simplu, trunchierea unui fişier se realizează prin redirectarea unei comenzi care nu afişează nimic la ieşirea standard. O astfel de comandă este : (comanda care nu face nimic):

```
razvan@anaconda:~/junk/scripting$ : > out.txt
```

Şi mai simplu, trunchierea se realizează folosind comanda de mai jos:

```
razvan@anaconda:~/junk/scripting$ > out.txt
```

Comanda este echivalentă cu cea de mai sus, cu absența comenzii :.

11.3.5.2 Înlănțuirea comenzilor

Shell-ul pune la dispoziția utilizatorului un set de operator care permit înlănțuirea diverselor comenzi pentru obținerea de noi funcționalități. Dintre aceştia, cel mai cunoscut este operatorul | (pipe) care redirectează ieşirea unei comenzi la intrarea alteia. Detalii despre acest operator se găsesc şi în capitolul 4.

Operatorul |

Un exemplu de utilizare al operatorului | este căutarea unei informații într-un fişier. O comandă uzuală este:

```
$ cat file.txt | grep keyword
```

Căutarea informației în fişier se realizează cu ajutorul comenzii `grep`, descrisă în secțiunea 11.5.2.

Un exemplu mai complex care utilizează o serie de comenzi prezentate în acest capitol este determinarea numărului de utilizatori unici care s-au autentificat în sistem într-o zi dată. Comenzile utilizate sunt `last`, `grep`, `cut`, `sort`, `uniq`, `wc`. Detalii despre aceste vor fi prezentate în secțiunile următoare.

Pentru început, vom folosi comanda last care afişează autentificările în sistem. Comanda `grep` este folosită pentru a extrage numai acele linii care conțin ziua dorită:

```
razvan@anaconda:~/junk/scripting$ last -30 | grep Mon
sergiu   pts/4        89.18.20.56      Mon Sep  3 23:36 - 23:46  (00:09)
razvan   pts/4        dhcp-204.cs.pub. Mon Sep  3 23:19 - 23:35  (00:15)
carpalex pts/0        86.121.140.140   Mon Sep  3 23:14 - 01:41  (02:27)
alexef   pts/0        89.120.196.59    Mon Sep  3 22:42 - 22:51  (00:08)
cojocar  pts/0        86.127.17.156    Mon Sep  3 19:50 - 19:51  (00:01)
ddvlad   pts/3        84.247.45.23     Mon Sep  3 16:00 - 16:01  (00:00)
root     pts/0        dhcp-204.cs.pub. Mon Sep  3 15:14 - 17:33  (02:18)
ddvlad   pts/0        84.247.45.23     Mon Sep  3 15:07 - 15:09  (00:01)
valentin pts/2        86.122.195.234   Mon Sep  3 13:47 - 02:43  (12:55)
alexef   pts/0        89.120.196.59    Mon Sep  3 13:30 - 15:04  (01:34)
```

```
razvan   pts/0        dhcp-204.cs.pub. Mon Sep  3 10:30 - 10:44  (00:13)
stefanb  pts/0        adsl-76-200-147- Mon Sep  3 10:05 - 10:17  (00:12)
stefanb  pts/0        adsl-76-200-147- Mon Sep  3 06:13 - 07:57  (01:43)
```

După obținerea autentificărilor în sistem vrem să extragem numai numele utilizatorilor. Pentru aceasta vom folosi comanda `cut`:

```
razvan@anaconda:~/junk/scripting$ last -30 | grep Mon | cut -d' ' -f1
sergiu
razvan
carpalex
alexef
cojocar
ddvlad
root
ddvlad
valentin
alexef
razvan
stefanb
stefanb
```

În acest moment avem numele utilizatorilor care s-au autentificat în sistem în ziua de Luni. Mai departe va trebui să extragem o singură instanță a numelor care se repetă. Drept urmare vom folosi utilitarele `sort` și `uniq`:

```
razvan@anaconda:~/junk/scripting$ last -30 | grep Mon | cut -d' ' -f1 | sort | uniq
alexef
carpalex
cojocar
ddvlad
razvan
root
sergiu
stefanb
valentin
```

Mai rămâne să afișăm numărul de autentificări. Acest lucru se realizează prin contorizarea liniilor obținute la pasul anterior, cu ajutorul utilitarului `wc`:

```
  razvan@anaconda:~/junk/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
sort | uniq | wc -l
  9
```

Atenție: Comanda de mai sus poate da rezultate incorecte în cazul în care `Mon` mai apare altundeva în rezultatul furnizat de last. Dacă ar exista un utilizator `Monica`, atunci liniile ce descriu autentificarea în sistem a utilizatorului `Monica` (chiar dacă autentificarea a avut sau nu loc în ziua de luni). Această problemă va fi corectată în secțiunea 11.5.7.

Operatorii ;, || și &&

Operatorii `;`, `||` și `&&` de înlănțuire a comenzilor țin cont de valoarea întoarsă de comanda anterioară. Astfel:

- `comm1 ; comm2` – comanda `comm2` se execută după execuția comenzii `comm1`
- `comm1 || comm2` – comanda `comm2` se execută în cazul în care `comm1` se întoarce cu o valoare nenulă
- `comm1 && comm2` – comanda `comm2` se execută în cazul în care `comm1` se încheie cu succes (întoarce `0`)

Operatorul `;` este echivalent cu execuția secvențială a fiecărei comenzi.

Operatorul `&&` este folosit când `comm2` depinde de execuția cu succes a comenzii `comm1`. O situație este compilarea din surse și instalarea unei aplicații:

```
# make && make install
```

11.4 Programarea shell

După prezentarea câtorva comenzi simple şi a operatorilor shell, putem pătrunde în aspectele de programare a unui shell script. După cum s-a precizat, shell-ul pune la dispoziția utilizatorului un limbaj de programare. Acesta conține, ca şi alte limbaje de programare, variabile, funcții, structuri de ciclare, instrucțiuni de decizie. Combinate cu celelalte facilități oferite de shell (utilizarea de comenzi deja implementate, înlănțuirea comenzilor, redirectarea), programarea shell devine un instrument puternic la îndemâna utilizatorului.

11.4.1 Variabile

O variabilă are asociat un nume şi o valoare. La fel ca în C, numele variabilelor este un şir de caractere ce poate conține litere, cifre sau underscore şi trebuie să înceapă cu literă sau underscore. Exemple de nume de variabile sunt `a`, `a0b`, `a_b`, `AxB_01_D`, etc.

Spre deosebire de limbaje precum C, variabilele în shell nu au un tip. Faptul că o variabilă este un număr sau un şir de caractere depinde de contextul în care este folosită. De asemenea, o variabilă poate fi definită orinde. În mod obişnuit, unei variabile îi este asociată o valoare în momentul definirii. Exemple de inițializări sunt:

```
a=0
b="alfa"
c="a08ss"
d=230
```

Un lucru important de reținut este fragilitatea programării shell şi constrângerile importante pe care le impune. Una dintre ele este faptul că nu poate exista spațiu înainte sau după semnul = din momentul inițializării unei variabile. Astfel, următoarele inițializări vor fi invalide:

```
a =0
b = "alfa"
d= 230
```

Valoarea unei variabile este referită cu ajutorul simbolului `$`. Astfel, dacă dorim afişarea valorii variabilelor definite mai sus folosim comanda:

```
razvan@anaconda:~$ echo $a $b $c $d
0 alfa a08ss 230
```

În consecință, dacă dorim inițializarea unei variabile la valoarea unei alteia vom folosi o construcție de forma:

```
razvan@anaconda:~$ echo $e
0
razvan@anaconda:~$ f="$b$c"
razvan@anaconda:~$ echo $f
alfaa08ss
```

În ultimul exemplu se observă că variabila `f` conține valorile concatenate ale celorlalte două variabile. De asemenea, se poate observa că ghilimelele nu schimbă sensul simbolului `$`.

Dacă se doreşte ca o variabilă să fie un rezultat al unei operații aritmetice cu o altă variabilă se foloseşte operatorul de expandare aritmetică descris în secțiunea 11.6.1.3:

```
razvan@anaconda:~$ a=4
razvan@anaconda:~$ c=3
razvan@anaconda:~$ echo $(($a * $c + 1))
13
```

O facilitate importantă a shell-ului este posibilitatea stocării ieşirii unei comenzi într-o variabilă. Acest lucru se realizează prin intermediul construcției `$(command)`. În exemplul de mai jos, reținem în variabila myusers numărul de utilizatori care au directorul de bază în `/home`:

```
razvan@anaconda:~$ local_users_num=$(cat /etc/passwd | grep /home | wc -l)
razvan@anaconda:~$ echo $local_users_num
52
```

Dacă vrem să contorizăm şi utilizatorul root, adăugăm `1` la acea variabilă:

```
razvan@anaconda:~$ total_local=$(($local_users_num + 1))
razvan@anaconda:~$ echo $total_local
53
```

***NOTĂ**: O greşeală frecventă în inițializarea unei variabile este folosirea de spații înainte sau după =. Evitați apariția unor astfel de greşeli în scripturile voastre.*

11.4.1.1 Câteva exemple

Afişarea ultimelor n adrese IP de la care s-au realizat conexiuni pentru un serviciu (inspecție de jurnale)

```
anaconda:/home/razvan/junk/scripting# cat -n service_ip.bash
     1  #!/bin/bash
     2
     3  ip_num=3
     4  log_file="/var/log/apache/access.log"
     5
     6  tail -n $ip_num $log_file | cut -d ' ' -f 1
anaconda:/home/razvan/junk/scripting# bash service_ip.bash
38.99.44.102
38.99.44.102
38.99.44.102
```

Afişarea memoriei totale ocupate de primele 3 procese din sistem

```
anaconda:/home/razvan/junk/scripting# cat -n mem_proc.bash
     1  #!/bin/bash
     2
     3  rss1=$(ps -e -o rss --sort=-rss | head -n 2 | tail -n 1)
     4  rss2=$(ps -e -o rss --sort=-rss | head -n 3 | tail -n 1)
     5  rss3=$(ps -e -o rss --sort=-rss | head -n 4 | tail -n 1)
     6
     7  total_rss_kb=$(($rss1 + $rss2 + $rss3))
     8
     9  echo "memorie totala ocupata de primele 3 procese: $total_rss_kb
KB"
anaconda:/home/razvan/junk/scripting# bash mem_proc.bash
memorie totala ocupata de primele 3 procese: 29088 KB
```

Adăugarea unui utilizator în sistem în mod neinteractiv

```
anaconda:/home/razvan/junk/scripting# cat -n my_add_user.sh
     1  #!/bin/bash
     2
     3  #
     4  # Add a new user; invoke all necesary script and command for
```

```
adding complete
        5  # user facilities
        6  #
        7
        8  username="newuser"
        9  email="razvand@gmail.com"
       10
       11  GROUP_NAME="students"
       12  GROUP_DIR="students"
       13  QUOTA="250"                 # size in megs
       14
       15  # add user
       16  useradd -m -d /home/$GROUP_DIR/$username -g $GROUP_NAME -s
/bin/bash $username
       17
       18  # create password
       19  password=$(pwgen -N 1)
       20
       21  # add password - non-interractive mode
       22  echo "$username:$password" | chpasswd
       23
       24  # set quota
       25  setquota $username $(($QUOTA * 1024)) $((($QUOTA + 10) * 1024))
$(($QUOTA * 10)) $((($QUOTA + 10) * 10)) -a
       26
       27  # send e-mail
       28  mail -s "New Account" $email <<EndOfMsg
       29  A new account has been created for you. Username is $username.
       30  Your password is $password. Please use passwd to change it.
       31
       32  Your qota limit is $QUOTA MB.
       33
       34  Have a nice day!
       35  EndOfMsg
       36
       37  exit 0
anaconda:/home/razvan/junk/scripting# ./my_add_user.sh
anaconda:/home/razvan/junk/scripting# cat /etc/passwd | tail -1
newuser:x:1051:1026::/home/students/newuser:/bin/bash
```

11.4.2 Caractere speciale shell

Shell-ul dispune de un set de caractere rezervate, care au roluri bine stabilite într-un script.

11.4.2.1 Caracterul blank (spațiu)

Caracterul blank este folosit pentru a separa argumentele unei comenzi de comandă şi între ele. Astfel, în exemplul

```
$ ls -l mydir
```

avem comanda ls cu două argumente: `-l` şi `mydir`. Se poate întâmpla să avem un argument care conține caractere blank. Spre exemplu, avem directorul `my dir`. Pentru afişarea conținutului acestui director, nu putem folosi comanda:

```
$ ls my dir
```

întrucât shell-ul ar considera două argumente transmise comenzii `ls`. Astfel, s-ar încerca afişarea conținutului directorului `my` şi apoi a directorului `dir`. Pentru a preîntâmpina acest efect, caracterul blank trebuie „citat", adică trebuie folosit ca un caracter obişnuit. Pentru aceasta există trei soluții:

- folosirea caracterului special ghilimele pentru „înglobarea" numelui de director ce conține blank:

```
$ ls "my dir"
```

- folosirea caracterului special apostrof, la fel cum s-a folosit ghilimele:

```
$ls 'my dir'
```

- folosirea caracterului special backslash, care dezactivează caracterul special de după el

```
$ ls my\ dir
```

Despre caracterele speciale ghilimele, apostrof şi backslash se va discuta în continuare.

11.4.2.2 Caracterul $ (dolar)

Unul dintre acestea este chiar caracterul `$` care poate fi folosit pentru expandarea unei variabile (a valorii acesteia), a unei comenzi sau expandare aritmetică.

Expandarea unei variabile se referă la determinarea valorii acesteia:

```
razvan@anaconda:~$ a=3
razvan@anaconda:~$ b=mesaj
razvan@anaconda:~$ echo $a $b
3 mesaj
```

Expandarea unei comenzi se refera la posibilitatea reținerii rezultatului rulării comenzii:

```
razvan@anaconda:~$ num_dirs=$(ls -l | wc -l)
razvan@anaconda:~$ echo $num_dirs
21
```

Expandare aritmetică se referă la executarea de calcule aritmetice. Fără folosirea operatorului de expandare aritmetică, operatorii aritmetici ar fi considerați caractere simple:

```
razvan@anaconda:~$ echo 2+3+4
2+3+4
razvan@anaconda:~$ echo $((2+3+4))
9
```

Mai multe informații despre caracterul `$` şi despre expandare sunt prezente în secțiunea 11.6.

Caracterul `$` este un caracter special. Acest lucru înseamnă că nu poate fi folosit cu semnificația de caracter direct când este urmat de un alt caracter care impune o expandare. Dacă dorim afişarea şirului `$a`, nu putem folosi comanda `echo $a`, pentru că ar afişa valoarea variabilei `a`:

```
razvan@anaconda:~$ echo $a
3
```

Pentru aceasta trebuie folosit caracterul backslash sau apostrof. **Atenție!** Folosirea ghilimelelor nu conduce la folosirea semnificației de caracter pentru `$`:

```
razvan@anaconda:~$ echo "$a"
3
razvan@anaconda:~$ echo '$a'
$a
razvan@anaconda:~$ echo \$a
$a
```

11.4.2.3 Caracterul " (ghilimele)

Un alt caracter special este " (ghilimele). Acest caracter este folosit pentru a defini şiruri de caractere:

```
$ a="sir"
$ b="acest sir"
```

Caracterul ghilimele este folosit pentru păstrarea semnificaţiei de caracter pentru caracterele speciale: `blank`, `(`, `)`, `&`, `|`, etc. Excepţia o constituie caracterul `$` care îşi păstrează semnificaţia în cazul expansiunii:

```
razvan@anaconda:~$ c=3
razvan@anaconda:~$ b=a
razvan@anaconda:~$ echo "c = $c; b = $b"
c = 3; b = a
```

Pentru afişarea unui caracter ghilimele putem folosi backslash sau apostrof:

```
razvan@anaconda:~$ echo "simbolul ghilimele (\")"
simbolul ghilimele (")
razvan@anaconda:~$ echo 'simbolul ghilimele (")'
simbolul ghilimele (")
```

11.4.2.4 Caracterul \ (backslash)

S-a observat, însă, că nu putem afişa caracterul `$`. Comanda de mai jos afişează valoarea variabilei `a`:

```
razvan@anaconda:~$ echo $a
3
```

Shell-ul interpretează caracterul `$` ca iniţiator al unei expandări. Dacă dorim afişarea şirului `$a`, nu putem folosi construcţia de mai jos, întrucât se va încerca expandarea variabilei a. Pentru a realiza acest lucru vom folosi caracterul \ (backslash). Acest caracter este caracterul de citarea şi are rolul de a păstra semnificaţia caracterului de după el. Citare înseamnă folosirea unui caracter cu semnificaţia literală.

Astfel, dacă dorim afişarea şirului `$a`, folosim comanda:

```
razvan@anaconda:~$ echo \$a
$a
```

Caracterul \ poate fi folosit şi pentru citarea caracterului ":

```
razvan@anaconda:~$ echo \"
"
```

Tot \ este folosit pentru afişarea \:

```
razvan@anaconda:~$ echo \\
\
```

11.4.2.5 Caracterul ' (apostrof)

Caracterul ' are un rol similar cu cel al caracterului ". Este folosit pentru descrierea de şiruri şi forţează caracterele speciale să-şi păstreze semnificaţia. Deosebirea între ' şi " este că

primul este mult mai puternic. Astfel, orice caractere speciale care apar între două caractere apostrof îşi păstrează semnificaţia:

```
razvan@anaconda:~$ echo ' " ) ; & $ \ '
" ) ; & $ \
```

11.4.2.6 Alte caractere speciale

Alte caractere cu rol special în shell sunt cele folosite de operatorii shell: `>`, `<`, `|`, `&`, `;`, `(`, `)` sau cele folosite pentru expresii regulate în shell: `{`, `}`, `*`, `+`, `?`, `[`, `]`. Toate aceste caractere îşi păstrează semnificaţia dacă sunt folosite între ghilimele sau apostroafe. Mai multe detalii despre caracterele folosite pentru expresii regulate în shell se regăsec în secţiunea 11.6.2.

Dacă, spre exemplu, un utilizator doreşte afişarea tuturor pachetelor instalate care încep cu litera `n`, va trebui să utilizeze comanda:

```
$ dpkg -l n*
```

Comanda de mai jos nu va funcţiona dacă există o intrare în directorul curent care începe cu litera `n` din cauza expandării expresiei regulate de către shell, fără a fi transmis către utilitarul `dpkg`:

```
razvan@anaconda:~/junk/scripting$ ls
hw.bash  new
razvan@anaconda:~/junk/scripting$ dpkg -l n*
No packages found matching new.
```

Soluţia este citarea caracterului `*`:

```
razvan@anaconda:~/junk/scripting$ dpkg -l 'n*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-============================================
ii  nano           2.0.2-1etch1   free Pico clone with some new features
un  nano-tiny      <none>         (no description available)
un  nas            <none>         (no description available)
un  nas-lib        <none>         (no description available)
pn  nautilus       <none>         (no description available)
pn  nautilus-cd-bu <none>         (no description available)
ii  nautilus-data  2.14.3-11      data files for nautilus
```

11.4.3 Instrucţiuni de decizie

Instrucţiunile de decizie folosite în scripturile shell sunt `if` şi `case`. O instrucţiune de decizie permite testarea unei condiţii şi executarea unei acţiuni sau a alteia conform condiţiei.

11.4.3.1 if

Vom analiza instrucţiunea `if` prin următorul exemplu: dacă utilizatorul local este `root` atunci se afişează un mesaj, dacă utilizatorul local este `razvan` se afişează un alt mesaj, iar dacă utilizatorul local este altcineva, se afişează un mesaj specific:

```
razvan@anaconda:~/junk/scripting$ cat -n if_user.bash
     1  #!/bin/bash
     2
     3  user=$(whoami)
     4
     5  if test $user = "root"; then
     6          echo "Bow before me for I am root."
     7  elif test $user = "razvan"; then
     8          echo "I am no humble user."
     9  else
    10          echo "I am but a humble user."
    11  fi
razvan@anaconda:~/junk/scripting$ bash if_user.bash
I am no humble user.
```

Opțiunea `-n` la `cat` afișează și numărul liniei. Sintaxa if este următoarea:

```
if conditie1; then
    actiune conditie1 indeplinita
elif conditie2; then
    actiune conditie2 indeplinita
else
    actiune conditie neindeplinita
fi
```

Cuvinte cheie predefinite sunt `if`, `the`, `elif`, `else`, `fi`. `conditie1`, respectiv `conditie2` sunt condiții care sunt testate. În cazul în care o conditie este îndeplinită se execută acțiunea asociată.

După cum se vede din exemplu (liniile 5 și 7), condițiile sunt exprimate cu ajutorul utilitarului `test`. Acesta interpretează condiția și întoarce `0` dacă este îndeplinită sau `1` dacă nu:

```
razvan@anaconda:~/junk/scripting$ test "alfa" = alfa
razvan@anaconda:~/junk/scripting$ echo $?
0
razvan@anaconda:~/junk/scripting$ test "alfa" = alf
razvan@anaconda:~/junk/scripting$ echo $?
1
```

Variabila `$?` este o variabilă predefinită care deține valoarea de retur a ultimei expresii.

Același efect poate fi obținut cu ajutorul operatorului de test al shell-ului `[ .. ]`:

```
razvan@anaconda:~/junk/scripting$ [ "alfa" = alfa ]
razvan@anaconda:~/junk/scripting$ echo $?
0
razvan@anaconda:~/junk/scripting$ [ "alfa" = alf ]
razvan@anaconda:~/junk/scripting$ echo $?
1
```

***ATENȚIE**: Este obligatoriu un caracter blank (spațiu) după [și înainte de] la operatorul de test de condiție din shell.*

Condiția folosită `if` nu trebuie să fie neapărat o condiție de test. Se poate folosi orice construcție care poate fi evaluată la un număr. Dacă acel număr este `0` condiția este îndeplinită și se execută acțiunea corespunzătoare, altfel nu. În exemplu următor, se afișează un mesaj corespunzător doar dacă utilizatorul indicat de variabila `user` are directorul de bază în `/home` (nu este utilizator de sistem):

```
razvan@anaconda:~/junk/scripting$ cat -n home_user.bash
     1  #!/bin/bash
     2
     3  user="mihai"
     4
     5  if grep "$user" /etc/passwd | grep "/home"; then
     6          echo "User $user is not homeless."
     7  else
     8          echo "User $user is homeless."
     9  fi
razvan@anaconda:~/junk/scripting$ bash home_user.bash
amihaiuc:x:1011:1011:Alex Mihaiuc,,,:/home/amihaiuc:/bin/bash
mihai:x:1014:1014:Mihai Dobrescu,,,:/home/mihai:/bin/bash
mihaif:x:1023:1023:Mihai Florian,,,:/home/mihaif:/bin/bash
User mihai is not homeless.
```

Se observă că if testează ieşirea comenzii compuse `grep "$user" /etc/passwd | grep "/home"` care verifică dacă există utilizatorul mihai în sistem şi dacă acest utilizator are directorul de bază în home (linia 5). Întrucât acesta există, afişează mesajul `"User mihai is not homeless."`.

Totuşi, comanda `grep` afişează şi liniile găsite, lucru care este deranjant în perspectiva funcționalității scriptului. Soluția este redirectarea ieşirii comenzii `grep` către `/dev/null`.

```
razvan@anaconda:~/junk/scripting$ cat -n home_user.bash
     1  #!/bin/bash
     2
     3  user="mihai"
     4
     5  if grep "mihai" /etc/passwd | grep "/home" > /dev/null ; then
     6          echo "User $user is not homeless."
     7  else
     8          echo "User $user is homeless."
     9  fi
razvan@anaconda:~/junk/scripting$ bash home_user.bash
User mihai is not homeless.
```

Condiții de test

Atât test cât şi operatorul de condiție shell `[ .. ]` analizează o condiție şi întorc `0` dacă este îndeplinită sau `1` dacă nu este îndeplinită. În exemplele prezentate de noi condiția a fost `sir1 = sir2`. Există mai multe condiții, o parte din care sunt prezentate în tabelul de mai jos:

operator	tip operand/operanzi	condiție verificată	utilizare
`-e`	1 fişier	fişierul există	`-e fisier`
`-f`	1 fişier	fişierul este fişier obişnuit (regular file)	`-f fisier`
`-s`	1 fişier	fişierul are dimensiune non-zero	`-s fisier`
`-d`	1 fişier	fişierul este director	`-d fisier`
`-r`	1 fişier	fişierul poate fi citit	`-r fisier`
`-w`	1 fişier	fişierul poate fi scris	`-w fisier`
`-x`	1 fişier	fişierul poate fi executat	`-x fisier`
`-O`	1 fişier	fişierul este deținul de utilizator	`-O fisier`
`-G`	1 fişier	grupul ce deține fişierul este şi grupul utilizatorului	`-G fisier`
`-nt`	2 fişiere	primul fişier este mai recent decât al doilea	`fisier1 -nt fisier2`

`-ot`	2 fişiere	primul fişier este mai puţin recent decât al doilea	`fisier1 -ot fisier2`
`-eq`	2 numere	numerele sunt egale	`numar1 -eq numar2`
`-ne`	2 numere	numerele nu sunt egalee	`numar1 -nq numar2`
`-gt`	2 numere	primul număr este mai mare decât al doilea	`numar1 -gt numar2`
`-ge`	2 numere	primul număr este mai mic sau egal cu al doilea	`numar1 -ge numar2`
`-lt`	2 numere	primul număr este mai mic decât al doilea	`numar1 -lt numar2`
`-le`	2 numere	primul număr este mai mic sau egal cu al doilea	`numar1 -le numar2`
`=`	2 şiruri de caractere	şirurile sunt identice	`sir1 = sir2`
`!=`	2 şiruri de caractere	şirurile nu sunt identice	`sir1 != sir2`
`-z`	1 şir de caractere	şirul este vid	`-z sir`

Tabelul 11.1 - Condiţii shell

Mai multe condiţii pot fi folosite simultan folosind operatorii `-a` şi `-o` cu sintaxele respectiv `conditie1 -a conditie2` şi `conditie1 -o conditie2`.

Câteva exemple de folosire a condiţiilor de mai sus sunt prezentate în continuare:

```
razvan@anaconda:~/junk/scripting$ test -f a.txt
razvan@anaconda:~/junk/scripting$ test -s a.txt
razvan@anaconda:~/junk/scripting$ test a.txt -ot b.txt
razvan@anaconda:~/junk/scripting$ test 3 -eq 4
razvan@anaconda:~/junk/scripting$ test -z ""
razvan@anaconda:~/junk/scripting$ test -z "a"
razvan@anaconda:~/junk/scripting$ test "a" = "b"
razvan@anaconda:~/junk/scripting$ test -f a.xt -a 3 -eq 4
```

Negarea unui condiţii se face cu ajutorul operatorului `!`:

```
  razvan@anaconda:~$ if ! test 4 -gt 5; then echo "4 nu e mai mare ca 5";
else echo "4 e mai mare ca 5"; fi
  4 nu e mai mare ca 5
```

Operatorul `((...))` de expansiune aritmetică poate fi folosit pentru evaluarea unor condiţii într-un format asemănător C:

```
razvan@anaconda:~/junk/scripting$ if ((1 < 4)); then echo "adevarat"; fi
adevarat
```

Mai multe detalii despre operatorul `((...))` se găsesc în secţiunea 11.6.1.3.

11.4.3.2 case

Instrucţiunea `case` permite selectarea între mai multe şabloane de şiruri de expresii regulate. În exemplul de mai jos se verifică dacă un şir începe cu o literă mică, o literă mare sau cu un număr:

```
razvan@anaconda:~/junk/scripting$ cat -n case_begin.bash
     1  #!/bin/bash
     2
     3  var="Alfa"
     4
     5  case $var in
     6          [a-z]*) echo "litera mica";;
     7          [A-Z]*) echo "litera mare";;
     8          [0-9]*) echo "cifra";;
     9          *) echo "altceva";;
    10  esac
razvan@anaconda:~/junk/scripting$ bash case_begin.bash
litera mare
```

Sintaxa pentru case, desprinsă şi din exemplu este:

```
case valoare_variabila in
   sablon1) actiuni;;
   sablon2) actiuni;;
   ...
esac
```

Cuvinte rezervate sunt `case`, `in`, `esac`. Se verifică dacă `valoare_variabila` se potriveşte cu unul dintre şabloanele ulterioare. În momentul găsirii unei potriviri se execută acţiunile asociate. Acţiunile asociate şablonului se încheie cu `;;` (două simboluri „punct şi virgulă").

În exemplul de mai sus, variabila `var` se iniţializează la şirul `„Alfa"` (linia 3). După aceasta se verifică dacă variabila începe cu literă mică (linia 6), cu literă mare (linia 7) sau cu cifră (linia 8). Şabloanele sunt date, de obicei, de expresii regulate. Mai multe detalii despre expresii regulate se găsesc în secţiunea 11.6.2.

Caracterul `*` înseamnă orice şi este folosit ca echivalent pentru cuvântul cheie `default` din C.

11.4.4 Cicluri în shell

Bash pune la dispoziţia utilizatorului trei instrucţiuni de ciclare cu funcţionalitate similară dar sintaxă diferită: `for`, `while` şi `do/until`. Vom exemplifica sintaxa şi modul de utilizare a acestor comenzi pe un exemplu simplu: calculul sumei primelor 10 numere naturale.

11.4.4.1 Instrucţiunea for

Scriptul shell pentru calculul sumei primelor 10 numere naturale este următorul:

```
razvan@anaconda:~/junk/scripting$ cat -n for10.bash
     1  #!/bin/bash
     2
     3  sum=0
     4  for i in 1 2 3 4 5 6 7 8 9 10; do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"
razvan@anaconda:~/junk/scripting$ bash for10.bash
Suma este: 55
```

După cum se vede, sintaxa `for` este

```
for variabila in lista; do
   actiuni
done
```

sau

```
for variabila in lista
do
    actiuni
done
```

`varibila` este folosita pentru a parcurge `lista` element cu element. Cuvinte rezervate sunt `for`, `in`, `do` şi `done`.

Se parcurg elementele listei (linia 4) şi se execută acţiunile precizate (linia 5). În cazul de faţă (linia 5), folosind operatorul de expansiune aritmetică `$((...))` se incrementează suma cu valoarea lui `i`.

Ca orice alte facilităţi shell, `for` poate fi folosit direct în linia de comandă:

```
razvan@anaconda:~/junk/scripting$ for i in 1 2 3 4; do echo $i; done
1
2
3
4
```

Acest lucru se poate extinde şi la scriptul de mai sus cu pierderea parţială a clarităţii:

```
razvan@anaconda:~/junk/scripting$ sum=0; for i in 1 2 3 4 5 6 7 8 9 10; do
sum=$(($sum+$i)); done; echo "Suma este: $sum"
Suma este: 55
```

sau:

```
razvan@anaconda:~$ sum=0
razvan@anaconda:~$ for i in 1 2 3 4 5 6 7 8 9 10; do
> sum=$(($sum + $i))
> done
razvan@anaconda:~$ echo $sum
55
```

Lista poate conţine orice fel de element, fie număr fie şir de caractere:

```
razvan@anaconda:~/junk/scripting$ for i in abc def ghi; do echo $i; done
abc
def
ghi
```

Separatorul implicit pentru listă este caracterul blank (spaţiu). Dacă un şir conţine blank, acesta trebuie citat:

```
razvan@anaconda:~/junk/scripting$ for i in "abc def" ghi; do echo $i; done
abc def
ghi
razvan@anaconda:~/junk/scripting$ for i in 'abc def' ghi; do echo $i; done
abc def
ghi
razvan@anaconda:~/junk/scripting$ for i in abc\ def ghi; do echo $i; done
abc def
ghi
```

Comanda seq

Revenind la calculul sumei primelor 10 numere naturale se observă lipsa de scalabilitate a specificării manuale a elementelor listei. Acest aspect poate fi automatizat prin folosirea comenzii `seq`. Aceasta permite generarea unei liste de numere într-un interval dat:

```
razvan@anaconda:~/junk/scripting$ seq 1 4
1
2
3
4
razvan@anaconda:~/junk/scripting$ seq 1 2 10
1
3
5
7
9
```

Cu două argumente, `seq` afişează elementele cuprinse între cele două limite. Cu trei argumente, argumentul din mijloc (al doilea) este folosit ca pas de incrementare.

Dacă, spre exemplu, am dori calculul sumei primelor 100 de numere naturale am folosi urmatorul script shell:

```
razvan@anaconda:~/junk/scripting$ cat -n for10_seq.bash
     1  #!/bin/bash
     2
     3  sum=0
     4  for i in $(seq 1 100); do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"
razvan@anaconda:~/junk/scripting$ bash for10_seq.bash
Suma este: 5050
```

O altă soluție este folosirea expandării aritmetice ca în exemplul de mai jos:

```
razvan@anaconda:~/junk/scripting$ cat -n for10_exp.bash
     1  #!/bin/bash
     2
     3  sum=0
     4  for ((i = 1; i <= 100; i++)); do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"
razvan@anaconda:~/junk/scripting$ bash for10_exp.bash
Suma este: 5050
```

Se observă similaritatea cu sintaxa C pentru instrucțiunea `for`.

Lucrul cu sistemul de fişiere

Folosind `for` putem parcurge intrările dintr-un director al sistemului de fişier. O emulare a comenzii `ls` este următorul script shell:

```
razvan@anaconda:~/junk/scripting$ cat -n for_ls.bash
     1  #!/bin/bash
     2
     3  n=0
     4  for i in *; do
     5          n=$(($n + 1))
     6          if test -f "$i"; then
     7                  echo "f - $i"
     8          elif test -d "$i"; then
     9                  echo "d - $i"
    10          else
    11                  echo "u - $i"
    12          fi
    13  done
    14
echo "total $n"
razvan@anaconda:~/junk/scripting$ bash for_ls.bash
f - do_until10.bash
f - echo.bash
f - for10.bash
f - for10_exp.bash
f - for10_seq.bash
f - for_ls.bash
f - here.bash
f - hw.bash
d - test_dir
f - while10.bash
total 10
```

Folosirea caracterului * (linia 4) înseamnă expandarea conținutului directorului curent într-o listă de intrări care este parcursă cu ajutorul variabilei `i`. Variabila `n` (linia 3) este folosită pentru contorizarea numărului de intrări (linia 5) folosind expandare aritmetică. Se recomandă folosirea ghilimelelor în lucrul cu fișierele (linia 6) pentru a preveni caractere cu rol special în shell (vezi 11.4.2).

11.4.4.2 Instrucțiunea while

Calculul sumei primelor 10 numere naturale folosind `while` se realizează cu scriptul de mai jos:

```
razvan@anaconda:~/junk/scripting$ cat -n while10.bash
     1  #!/bin/bash
     2
     3  i=1
     4  n=10
     5  sum=0
     6
     7  while test $i -le $n; do
     8          sum=$(($sum + $i))
     9          ((i++))
    10  done
    11
    12  echo "Suma este: $sum"
razvan@anaconda:~/junk/scripting$ bash while10.bash
Suma este: 55
```

Sintaxa `while` este

```
while conditie; do
    actiuni
done
```

`conditie` urmează aceleaşi reguli ca în cazul `if`. Exemplul prezentat seamănă foarte mult cu un program C: iniţializări (liniile 2, 3, 4), condiţie (linia 7), acţiuni (linia 8), incrementare (linia 9). Se poate observa că incrementarea lui `i` (linia 9) s-a realizat prin intermediul operatorului de expandare aritmetică.

Comanda `while` este folosită cu precădere în momentul în care condiţia este dată de valoarea de retur a unei alte comenzi (de multe ori `read`). O emulare a comenzii cat cu opţiunea `-n` este exemplificată în scriptul următor:

```
razvan@anaconda:~/junk/scripting$ cat -n while_cat.bash
     1  #!/bin/bash
     2
     3  i=1
     4  cat for10.bash | while read a; do
     5          echo -e "$i:\t$a"
     6          i=$(($i + 1))
     7  done
     8
razvan@anaconda:~/junk/scripting$ bash while_cat.bash
1:      #!/bin/bash
2:
3:      sum=0
4:      for i in 1 2 3 4 5 6 7 8 9 10; do
5:      sum=$(($sum + $i))
6:      done
7:
8:      echo "Suma este: $sum"
```

Se observă că se citeşte în variabila a câte o linie din fişiereul `for10.bash` până la sfârşitul fişierului.

11.4.4.3 Instrucţiunea until

Comanda `until` este asemănătoare cu `while`, doar că în acest caz condiţia nu este cea de continuare a acţiunilor, ci este cea de oprire. Datorită similarităţii nu vom insista; scriptul shell care calculează suma primelor 10 numere naturale este prezentat în continuare:

```
razvan@anaconda:~/junk/scripting$ cat -n until10.bash
     1  #!/bin/bash
     2
     3  i=1
     4  sum=0
     5  n=10
     6
     7  until test $i -gt $n; do
     8          sum=$(($sum + $i))
     9          i=$(($i + 1))
    10  done
    11
    12  echo "Suma este: $sum"
razvan@anaconda:~/junk/scripting$ bash until10.bash
Suma este: 55
```

***NOTĂ**: Exemplele prezentate în secţiunea **Cicluri în shell** sunt pur academice. Nu există nici un motiv întemeiat pentru a calcula suma numerelor naturale sau a emula comenzi existente folosind un script shell. Rolul unui script shell este acela de a folosi cele mai potrivite utilitare existente pentru a rezolva o problemă.*

11.5 Filtre de text

Utilitarele de filtrare a textului primesc la intrare un fişier text şi oferă la ieşire o formă prelucrată a acestuia. Prelucrarea poate impune schimbarea ordinii elementelor, selecţia anumitor elemente (linii, coloane), substituţia unor elemente (linii, cuvinte, caractere), adăugarea de noi elemente (număr de linie, etc.) şi alte operaţii. Aceste utilitare sunt, de obicei, folosite în tandem cu instrucţiuni de decizie sau de ciclu shell pentru obţinerea unor informaţii ce pot fi utilizate ulterior.

11.5.1 cat, tac, nl

Utilitarul principal în prelucrarea de şiruri este `cat`. `cat` este folosit pentru afişarea unui fişier şi este, de multe ori, prima comandă dintr-o înlănţuire de comenzi care folosesc `|`. După cum s-a precizat în secţiunile anterioare, o opţiune utilă a `cat` este `-n` care afişează şi numărul liniei:

```
razvan@anaconda:~/junk/scripting$ cat -n nume.txt
     1  sorin
     2  stefania
     3  mihaela
     4  florin
     5  codrin
     6  lucian
     7  razvan
     8  tavi
```

Utilitarul `tac` (`cat` inversat) este folosit pentru afişarea unui fişier cu liniile inversate (prima linie prima):

```
razvan@anaconda:~/junk/scripting$ tac nume.txt
tavi
razvan
lucian
codrin
florin
mihaela
stefania
sorin
```

Utilitarul `nl` este folosit pentru a afişa liniile împreună cu numărul lor:

```
razvan@anaconda:~/junk/scripting$ nl for10_exp.bash
     1  #!/bin/bash

     2  sum=0
     3  for ((i = 1; i <= 100; i++)); do
     4          sum=$(($sum + $i))
     5  done

     6  echo "Suma este: $sum"
```

Deosebirea faţă de `cat -n` este faptul că `nl` nu consideră liniile goale. De asemenea, `nl` are un set de opţiuni care includ paginarea, considerare de secţiuni, etc. care depăşesc aria de cuprindere a acestei cărţi.

11.5.2 sort, uniq

Comanda `sort` este folosită pentru sortarea liniilor primite la intrare. În exemplul de mai jos, se realizează sortarea liniilor din fişierul `nume.txt`:

```
razvan@anaconda:~/junk/scripting$ cat nume.txt
sorin
stefania
mihaela
florin
codrin
lucian
razvan
tavi
razvan@anaconda:~/junk/scripting$ sort nume.txt
codrin
florin
lucian
mihaela
razvan
sorin
stefania
tavi
```

Se poate întâmpla ca un nume să apară de două ori şi să avem nevoie doar de o singură apariţie în ieşire. În această situaţie trebuie folosit utilitarul `uniq` sau opţiunea `-u` la `sort`:

```
razvan@anaconda:~/junk/scripting$ sort nume2.txt
alex
alina
alina
cristi
cristi
cristi
razvan
razvan
razvan@anaconda:~/junk/scripting$ sort nume2.txt | uniq
alex
alina
cristi
razvan
razvan@anaconda:~/junk/scripting$ sort -u nume2.txt
alex
alina
cristi
razvan
```

Un exemplu util a fost prezentat în secţiunea 11.3.3.2, Operatorul. În acel exemplu se dorea aflarea numărului de utilizatori care s-au autentificat în sistem într-o zi dată. Comanda utilizată a fost:

```
razvan@anaconda:~/junk/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
sort | uniq | wc -l
9
```

Opţiuni utile sort

Dintre opţiunile utile ale comenzii sort amintim:

- `-u`: elimină duplicatele după sortare;
- `-r`: sortare inversă (în ordine descrescătoare);

- `-t`: specificarea separatorului folosit pentru sortare; implicit, sortarea se realizează după primul câmp;
- `-n`: sortare numerică; implicit sortarea este alfanumerică, după cum se poate vedea în exemplul de mai jos:

```
razvan@anaconda:~/junk/scripting$ sort numere.txt
-3
10
100
12
15
2101
28
492
5
razvan@anaconda:~/junk/scripting$ sort -n numere.txt
-3
5
10
12
15
28
100
492
2101
```

Sortare avansată

Uneori poate apărea situația în care dorim sortarea după o coloană a fișierului de intrare. Presupunem că avem o miniagendă cu persoane în care intrările sunt în forma prenume, nume, număr de telefon și adresă de e-mail și dorim sortarea după nume (adică după a doua coloană). Pentru aceasta vom folosi comanda:

```
razvan@anaconda:~/junk/scripting$ cat nume3.txt
Adrian,Munteanu,0711 111 111,am@ex.com
Ilinca,Zafiu,0711 111 112,iz@ex.com
Andreea,Popa,0711 111 113,ap@ex.com
Dan,Badea,0711 111 114,db@ex.com
razvan@anaconda:~/junk/scripting$ sort -t , -k 2,2 nume3.txt
Dan,Badea,0711 111 114,db@ex.com
Adrian,Munteanu,0711 111 111,am@ex.com
Andreea,Popa,0711 111 113,ap@ex.com
Ilinca,Zafiu,0711 111 112,iz@ex.com
```

Am folosit argumentul `-t` pentru a specifica virgula ca separator. Pentru sortarea după o cheie specifică se folosește argumentul `-k`. Acesta specifică de la ce câmp se începe sortarea (aici este vorba de al doilea câmp) și cu ce câmp se termină (în cazul de față tot 2: `-k 2,2`). În caz de egalitate, se poate alege un nou câmp de sortare prin specificarea unui nou argument `-k`.

11.5.3 head, tail

Comenzile `head` și `tail` sunt utile pentru a reține primele sau ultimele linii dintr-un fișier primit la intrare. Implicit se rețin primele sau ultimele 10 linii. Exemple de utilizare sunt prezentate în continuare:

```
razvan@anaconda:~/junk/scripting$ head /etc/passwd
razvan@anaconda:~/junk/scripting$ tail /etc/passwd
razvan@anaconda:~/junk/scripting$ head -n 20 /etc/passwd
razvan@anaconda:~/junk/scripting$ tail -n 20 /etc/passwd
```

Comenzile specificate afişează respectiv: primele 10 linii din fişierul `/etc/passwd`, ultimele 10 linii, primele 20 de linii, ultimele 20 de linii.

Comanda tail este utilă pentru inspecţia jurnalelor (log-urilor). Jurnalele sunt de obicei fişiere de dimensiune mare şi se doreşte vizualizarea ultimelor intrări.

O opţiune utilă a comenzii tail este `-f`. Aceasta permite vizualizarea conţinutului unui fişier în timp real în sensul că aşteaptă scrierea unor noi linii în fişier şi afişarea acestora pe măsură ce acestea sunt adăugate:

```
anaconda:~# tail -f /var/log/apache/access.log
```

Întreruperea rulării comenzii se încheie prin apăsarea CTRL-C (transmiterea semnalului SIGINT).

11.5.4 cut

Comanda `cut` este utilă pentru a extrage coloane dintr-un fişier format corespunzător. Spre exemplu, dacă folosim fişierul prezentat în secţiunea anterioară şi vrem afişarea prenumelui şi a adresei de e-mail vom folosi comanda:

```
razvan@anaconda:~/junk/scripting$ cut -d ',' -f 1,4 < nume3.txt
Adrian,am@ex.com
Ilinca,iz@ex.com
Andreea,ap@ex.com
Dan,db@ex.com
```

Cele două opţiuni cele mai importante ale comenzii `cut` sunt `-d` pentru a specifica delimitatorul şi `-f` pentru a preciza câmpurile (coloanele) care se doresc extrase. În exemplul de mai sus, separatorul a fost virgulă şi s-au reţinut câmpurile (coloanele) 1 şi 4 corespunzătoare prenumelui şi adresei de e-mail.

Dacă se doreşte extragerea numelui de utilizator şi al directorului de bază pentru utilizatorii care au directorul de bază în `/home`, vom folosi următoarea comandă:

```
razvan@anaconda:~/junk/scripting$ cat /etc/passwd | grep /home | cut -d ':' -f1,6
ftp:/home/ftp
george:/home/students/george
andreic:/home/students/andreic
iepurasu:/home/students/iepurasu
valentin:/home/students/valentin
dhartescu:/home/students/dhartescu
ciconaru:/home/students/ciconaru
```

Comanda de mai sus extrage utilizatorii care au directorul de bază în /home (cat /etc/passwd | grep /home), după care extrage numele de utilizator şi directorul de bază asociat (cut –d ‚:' –f 1,6). Separatorul de câmp în /etc/passwd este :, iar indexurile de câmp corespunzătoare numelui de utilizator şi directorului de bază sunt 1 şi 6.

11.5.5 tr

Utilitarul `tr` (transliterate) este folosit pentru translatarea la nivel de caracter a informaţiilor de la intrare. Acţiuni posibile care pot fi efectuate cu ajutorul comenzii `tr` sunt:

- translatarea caracterelor,
- eliminarea caracterelor care se repetă (squeeze),
- ştergerea de caractere.

- Exemple de translatarea caracterelor sunt:

```
razvan@anaconda:~/junk/scripting$ tr a b <<<"acadaeaf"
bcbdbebf
razvan@anaconda:~/junk/scripting$ tr acd xyz <<<"acadaeaf"
xyxzxexf
razvan@anaconda:~/junk/scripting$ tr a-z A-Z <<<"acadaeaf"
ACADAEAF
razvan@anaconda:~/junk/scripting$ tr '[:lower:]' '[:upper:]' <<<"acadaeaf"
ACADAEAF
```

În primul exemplu se translatează caracterul `a` în caracterul `b`. Şirul este transmis sub formă de „here string". În cel de-al doilea exemplu, avem un set de intrare (`acd`) şi un set de ieşire (`xyz`). Translatarea se face între primul element din setul de intrare şi primul element din setul de ieşire, al doilea element din setul de intrare şi al doilea element din setul de ieşire, etc. Al treilea şi al patrulea exemplu translatează literele mari în litere mici. `a-z` simbolizează literele de la `a` la `z`. De asemenea, expresiile `[:lower:]` şi `[:upper:]` reprezintă, respectiv, literele mici şi literele mari din alfabet. Utilitarul `tr` foloseşte mai multe astfel de expresii. Mai multe detalii puteţi afla din pagina de manual sau info (`man tr`, `info coreutils`).

Exemple de eliminare a caracterelor care se repetă sunt următoarele:

```
razvan@anaconda:~/junk/scripting$ tr -s ' ' <<< "a    b    c"
a b c
razvan@anaconda:~/junk/scripting$ tr -s a-z A-Z <<< "aa    bb    cc"
A    B    C
```

În cazul primului exemplu, caracterul spaţiu este eliminat până la o singură apariţie. Opţiunea `-s` este folosită pentru a specifica acţiunea de eliminare a duplicatelor unui caracter (squeeze).

În cel de-al doilea exemplu se foloseşte simultan operaţiunea de translatare şi de eliminare de duplicate pe şirul de intrare `"aa  bb  cc"`. Rezultatul este transformarea literelor mici în litere mari simultan cu eliminarea duplicatelor acestora.

Ştergerea caracterelor se face cu ajutorul opţiunii `-d`:

```
razvan@anaconda:~/junk/scripting$ tr -d 'a' <<< 'abcdcba'
bcdcb
razvan@anaconda:~/junk/scripting$ tr -ds 'a' 'bcd' <<< 'aabbccddccbbaa'
bcdcb
```

Opţiunea `-d` primeşte un set de caractere care sunt eliminate. Opţiunea `-d` poate fi completată de opţiunea `-s`. În cel de-al doilea exemplu se şterge caracterul a din şirul de intrare iar celorlalte caractere le sunt eliminate duplicatele.

11.5.6 wc

Utilitarul `wc` (word count) permite contorizarea numărului de linii, de caractere sau de cuvinte dintr-un fişier text. La o folosire fără opţiuni, `wc` afişează atât numărul de linii cât şi numărul de cuvinte şi caractere:

```
razvan@anaconda:~/junk/scripting$ wc < for10.bash
  8  24 102
```

Pentru afişarea numai a numărului de linii, de cuvinte sau de caractere, se folosesc, respectiv, opţiunile `-l`, `-w`, `-c`:

```
razvan@anaconda:~/junk/scripting$ wc -l < for10.bash
8
razvan@anaconda:~/junk/scripting$ wc -w < for10.bash
24
razvan@anaconda:~/junk/scripting$ wc -c < for10.bash
102
```

11.5.7 Contorizare număr utilizatori autentificați în sistem – rezolvare corectă

Folosind cunoştinţele de până acum, se poate rezolva corect problema propusă în secţiunea 11.3.3.2 – contorizarea numărului de utilizatori autentificaţi în ziua de luni în sistem. Soluţia este prezentată în continuare:

```
razvan@anaconda:~/junk/scripting$ cat -n count_login.bash
     1  #!/bin/bash
     2
     3  user_day="Monday"
     4
     5  case $user_day in
     6          "Monday") day_last="Mon";;
     7          "Tuesday") day_last="Tue";;
     8          "Wednesday") day_last="Wed";;
     9          "Thursday") day_last="Thu";;
    10          "Friday") day_last="Fri";;
    11          "Saturday") day_last="Sat";;
    12          "Sunday") day_last="Sun";;
    13          *) echo "Zi invalida"; exit 1;;
    14  esac
    15
    16  last | head -n -2 | while read user term ip day other; do
    17          if test $day = $day_last; then
    18                  echo "$user";
    19          fi
    20  done | sort -u | wc -l
razvan@anaconda:~/junk/scripting$ bash count_login.bash
9
```

Linia 3 defineşte variabila `user_day` folosită pentru a reţine ziua despre care dorim informaţii (generalizat). Aceasta va fi dată în formatul complet pentru a avantaja utilizatorul. Liniile 5-14 se ocupă de traducerea zilei în formatul `last`; ziua în formatul last (prescurtat) este reţinută în variabila `day_last`.

Liniile 16-20 reprezintă, de fapt, o comandă înlănţuită. Componentele lanţului de comenzii sunt:

- `last` – afişează informaţii despre utilizatorii autentificaţi în sistem;
- `head -n -2` – reţine primele linii de la intrare, mai puţin ultimele două; este nevoie de acest lucru deoarece ultimele două linii date la ieşire de `last` nu sunt utile;
- ciclul `while` este folosit pentru a citi, folosind `read`, diversele câmpuri ale fiecărei linii date de `last`: numele de utilizator, terminalul asociat, adresa IP de conectare, ziua în care s-a realizat autentificarea; variabila `other`, fiind ultima variabilă, reţine restul de informaţii până la sfârşitul liniei (informaţii care nu interesează); dacă ziua corespunde zilei reţinute în variabila `day_last` atunci se afişează la ieşire numele de utilizator;
- `sort -u` – sortează şi reţine în mod unic liniile primite la intrare; în cazul nostru aceste linii sunt chiar cele afişate de ciclul `while`, adică utilizatorii care s-au autentificat în sistem în ziua dată de variabila `user_day`;

- `wc -l` – contorizează numărului de linii furnizat de `sort -u`; rezultatul este numărul de utilizatori distincți care s-au autentificat în sistem în ziua de dată de variabila `user_day`.

11.5.8 grep

Utilitarul `grep` şi variantele sale (`egrep`, `fgrep`) sunt utilitarele de bază în prelucrarea fişierelor text. Acest utilitar permite selectarea anumitor linii dintr-un fişier text pe baza unei expresii regulate transmise ca argument. `grep` primeşte ca parametru şirul de căutare şi, eventual, fişierul în care se face căutarea. Dacă dorim să vedem informații despre autentificarea utilizatorului `adrian` în sistem vom folosi comanda:

```
razvan@anaconda:~/junk/scripting$ last | grep adrian
adrian   pts/2        92.80.182.158    Sun Sep  2 22:30 - 22:38  (00:07)
adrian   pts/2        92.80.149.10     Sun Sep  2 09:10 - 09:29  (00:18)
adrian   pts/0        92.80.149.10     Sun Sep  2 09:06 - 09:29  (00:23)
```

Dacă dorim să afişăm informații despre utilizatorii din sistem al căror nume începe cu litera `a` vom folosi comanda:

```
razvan@anaconda:~/junk/scripting$ cat /etc/passwd | grep '^a'
alexpoke:x:1003:1003:Poke Alexandu,,,,A & C:/home/alexpoke:/bin/bash
adrian:x:1009:1009:Adrian Nistor,,,:/home/adrian:/bin/bash
amihaiuc:x:1011:1011:Alex Mihaiuc,,,:/home/amihaiuc:/bin/bash
alina:x:1016:1016:Alina Deaconescu,,,:/home/alina:/bin/bash
andrewbwm:x:1031:1026:Andrei Buhaiu:/home/students/andrewbwm:/bin/bash
alexef:x:1032:1026:Alex Eftimie:/home/students/alexef:/bin/bash
andreic:x:1045:1026:Andrei Cibotaru:/home/students/andreic:/bin/bash
```

În cazul de față, pentru obținerea utilizatorilor al căror nume începe cu litera `a` se foloseşte expresia regulată `^a`. Aceasta va însemna selectarea acelor linii care încep cu litera a (caracterul `^` este special şi simbolizează început de linie). Mai multe detalii despre expresii regulate `grep` se găsesc în secțiunea următoare.

`grep` are un set bogat de opțiuni. O parte din cele mai importante sunt prezentate în continuare pentru fişierul `grep_test.txt`:

```
razvan@anaconda:~/junk/scripting$ cat grep_test.txt
alfa beta
AlFa BeTa
Beta gamma
beta delta
deltadd epsilon
```

- opțiunea `-i` ignoră literele mari (ignore case):

```
razvan@anaconda:~/junk/scripting$ grep -i "alfa" < grep_test.txt
alfa beta
AlFa BeTa
```

- opțiunea `-v` inversează căutarea (afişează liniile care nu conțin expresie de căutare):

```
razvan@anaconda:~/junk/scripting$ grep -v "alfa" < grep_test.txt
AlFa BeTa
Beta gamma
beta delta
deltadd epsilon
razvan@anaconda:~/junk/scripting$ grep -v -i "alfa" < grep_test.txt
Beta gamma
beta delta
deltadd epsilon
```

- opțiunea `-n` afişează numărul liniei:

```
razvan@anaconda:~/junk/scripting$ grep -n "alfa" < grep_test.txt
1:alfa beta
```

- opțiunea `-w` este folosită pentru căutarea unui cuvânt întreg:

```
razvan@anaconda:~/junk/scripting$ grep -w "delta" < grep_test.txt
beta delta
```

- opțiunea `-r` face căutare recursivă în structura de directoare:

```
razvan@anaconda:~/junk/scripting$ grep -r "for" .
./for_ls.bash:for i in *; do
./while_cat.bash:cat for10.bash | while read a; do
./for10.bash:for i in 1 2 3 4 5 6 7 8 9 10; do
./for10_seq.bash:for i in $(seq 1 100); do
./for10_exp.bash:for ((i = 1; i <= 100; i++)); do
./if_user.bash: echo "Bow before me for I am root."
```

Rezultatul dat de utilizarea acestei opțiuni prezintă fișierele în care s-a găsit acel șir/expresie regulată și linia/liniile complete din fișiere.

11.5.8.1 Expresii regulate grep

După cum s-a observat, `grep` folosește un șir de căutare care poate avea forma unei expresii regulate.

O expresie regulată este un șir de caractere utilizat pentru a se potrivi cu un alte șiruri de caractere conform unor reguli de sintaxa bine precizate. O expresie regulată conține caractere simple și caractere cu semnificație specială. O parte din caracterele/grupările de caractere cu semnificație specială sunt prezentate în continuare:

- Caracterele `[]` sunt folosite pentru a defini un set de caractere de potrivit; exemple:
  - `[abcd]` înseamnă potrivire cu oricare din caracterele `a`, `b`, `c`, `d`;
  - `[a-z]` înseamnă potrivire cu oricare minusculă;
  - `[0-9a-zA-Z]` înseamnă potrivire cu orice caracter alfanumeric;
  - `[^A-Z]` înseamnă potrivire cu orice caracter mai puțin majusculele.
- Caracterul `.` înseamnă orice caracter;
- Caracterul `^` în afara unui set de caractere înseamnă potrivire cu începutul liniei;
- Caracterul `$` înseamnă potrivire cu sfârșitul liniei;
- Simbolurile `\<`, respectiv `\>` sunt folosite pentru a se potrivi cu șirul vid de la început și sfârșit de cuvânt;
- Simbolul `?` înseamnă potrivire de cel mult o dată a caracterului/grupului de caractere dinainte;
- Simbolul `*` înseamnă potrivire de zero sau mai multe ori a caracterului/grupului de caractere dinainte;
- Simbolul `+` înseamnă potrivire de o dată sau mai multe ori a caracterului/grupului de caractere dinainte;
- `{n}` înseamnă potrivire de `n` ori;
- `{n,}` înseamnă potrivire de cel puțin `n` ori;
- `{n,m}` înseamnă potrivire de cel puțin `n` ori și cel mult `m` ori.

Caracterele/grupurile de caractere prezentate mai sus pot fi concatenate pentru generarea unei expresii regulate complexe.

> ***NOTĂ:*** *Informații complete despre comenzile de filtrare de text folosite în shell scripting găsiți în paginile info asociate (**info coreutils**).*

11.6 Comenzi de lucru cu fişiere

Comenzile de lucru cu fişiere sunt comenzi care permit căutarea în sistemul de fişiere şi executarea unor acţiuni pentru fişierele căutate. Comenzile prezentate în acest capitol sunt `xargs`, `locate` şi `find`.

11.6.1 xargs

Comanda `xargs` permite transmiterea de argumente unei alte comenzi, argumente ce pot fi trimise de la intrarea standard. Spre exemplu, dacă avem un fişier ce conţine un set de fişiere pe care dorim să le ştergem, vom folosi comanda:

```
$ xargs rm < file.txt
```

ca în exemplul de mai jos

```
razvan@anaconda:~/junk/scripting$ cat file.txt
tmp1.txt
tmp2.txt
tmp3.txt
razvan@anaconda:~/junk/scripting$ ls tmp*
tmp1.txt  tmp2.txt  tmp3.txt
razvan@anaconda:~/junk/scripting$ xargs rm < file.txt
razvan@anaconda:~/junk/scripting$ ls tmp*
ls: tmp*: No such file or directory
```

În exemplu, fişierul `file` conţinea trei nume de fişier: `tmp1.txt`, `tmp2.txt`, `tmp3.txt`. Folosind `xargs` se invocă `rm`. Argumentele pentru `rm` sunt citite de la intrarea standard în care a fost redirectat fişierul `file.txt`. Rezultatul este ştergerea fişierelor `tmp1.txt`, `tmp2.txt` şi `tmp3.txt` precizate în fişierul `file.txt`.

Dacă un utilizator doreşte afişarea fişierelor ce conţin şirul for va utiliza comanda:

```
$ grep -l 'for' *
```

sau

```
$ ls | xargs grep -l 'for'
```

La fel, dacă se doreşte generarea unei liste a utilizatorilor în sistem se poate utiliza comanda:

```
  razvan@anaconda:~/junk/scripting$ cut -d ':' -f 1 < /etc/passwd | sort |
tr -s '\n' ' '
  Debian-exim adrian alexef alexpoke alina amihaiuc [...]
```

sau

```
  razvan@anaconda:~/junk/scripting$ cut -d ':' -f 1 < /etc/passwd | sort |
xargs echo
  Debian-exim adrian alexef alexpoke alina amihaiuc [...]
```

Opţiuni utile pentru `xargs` sunt:

- `-d` este folosită pentru a specifica delimitatorul de parametri; implicit acesta este orice caracter alb (whitespace); în exemplul de mai jos se şterg fişierele `tmp1.txt`, tmp2.txt şi tmp3.txt:

```
razvan@anaconda:~/junk/scripting$ cat file2.txt
tmp1.txt,tmp2.txt,tmp3.txt
razvan@anaconda:~/junk/scripting$ ls tmp*
tmp1.txt  tmp2.txt  tmp3.txt
razvan@anaconda:~/junk/scripting$ xargs -d ',' rm  < file2.txt
rm: cannot remove `\n': No such file or directory
```

Ultima linie apare ca urmare a faptului că se interpretează caracterul newline (`\n`) ca un argument posibil.

- `-0` impune folosirea ca delimitator de argument a nul-terminatorului de sir (`\0`); acest lucru este util în momentul în care parametrii conţin caractere speciale (blank, backslash, etc.); este folosit, de obicei, împreună cu argumentul `-print0` la `find` [vezi 11.6.3]
- `-I` permite folosirea unui şir care va fi substituit pe parcursul comenzii cu argumentul primit la intrare; în următorul exemplu, fişierele primite la intrare sunt copiate cu extensia `.bkup` în `/tmp`:

```
razvan@anaconda:~/junk/scripting$ cat file3.txt
for10.bash
for10_seq.bash
for10_exp.bash
razvan@anaconda:~/junk/scripting$ xargs -I abc cp abc /tmp/abc.bkup < file3.txt
razvan@anaconda:~/junk/scripting$ ls /tmp/*.bkup
/tmp/for10.bash.bkup  /tmp/for10_exp.bash.bkup  /tmp/for10_seq.bash.bkup
```

Se observă că s-a folosit şirul `abc` ca şir de substituit (prin intermediul opţiunii `-I`). La apariţia acestui şir în comanda folosită de `xargs`, acesta a fost înlocuit cu parametrul de la intrare (în cazul nostru fiecare din cele trei fişiere). Astfel, comanda

```
$ xargs -I abc cp abc /tmp/abc.bkup
```

este echivalentă cu

```
$ cp for10.bash for10.bash.bkup
$ cp for10.bash for10_seq.bash.bkup
$ cp for10.bash for10_exp.bash.bkup
```

11.6.2 locate

Comanda `locate` permite localizarea de fişiere al căror nume corespunde unui anumit şablon transmis ca parametru. Comanda locate foloseşte o bază de date (sau un set de bază de date) construită cu ajutorul comenzii `updatedb`.

Un exemplu este prezentat în continuare:

```
razvan@anaconda:~/junk/scripting$ time locate strace
/usr/bin/strace
/usr/share/doc/strace
/usr/share/doc/strace/changelog.Debian.gz
/usr/share/doc/strace/changelog.gz
/usr/share/doc/strace/copyright
/usr/share/doc/strace/TODO.gz
/usr/share/man/man1/strace.1.gz
real    0m0.288s
user    0m0.284s
sys     0m0.004s
```

După cum se vede comanda `locate` este foarte rapidă întrucât foloseşte baza de date generată/actualizată anterior de `updatedb`. Întrucât este posibil ca anumite fişiere să fi fost şterse de la ultima actualizare, se poate folosi opţiunea `-e`:

```
razvan@anaconda:~/junk/scripting$ locate -e strace
/usr/bin/strace
/usr/share/doc/strace
/usr/share/doc/strace/changelog.Debian.gz
/usr/share/doc/strace/changelog.gz
[...]
```

Comanda `locate` este de multe ori folosită împreună cu `xargs`:

```
razvan@anaconda:~/junk/scripting$ locate -e strace | xargs file
/usr/bin/strace:                                ELF 32-bit LSB executable,
Intel 80386, version 1 (SYSV), for GNU/Linux 2.4.1, dynamically linked (uses
shared libs), for GNU/Linux 2.4.1, stripped
/usr/share/doc/strace:                          directory
/usr/share/doc/strace/changelog.Debian.gz: gzip compressed data, was
"changelog.Debian", from Unix, last modified: Fri Oct 27 00:22:12 2006, max
compression
[...]
```

11.6.3 find

Comanda `find` este principala comanda utilizată pentru căutarea în sistemul local de fişiere. Comanda realizează o căutare recursivă în structura de directoare şi descoperă acele intrări în sistemul de fişiere care satisfac o anumită condiție. Suplimentar, comanda `find` permite executarea unei acțiuni specifice.

Sintaxa `find` este mai complexă decât a altor comenzi şi o vom prezenta după următorul exemplu:

```
razvan@anaconda:~$ find /usr/include/ -type f -name '*term*.h' -print
/usr/include/asm-generic/termios.h
/usr/include/linux/termios.h
/usr/include/asm/termbits.h
/usr/include/asm/termios.h
/usr/include/bits/termios.h
/usr/include/sys/termios.h
/usr/include/termio.h
/usr/include/termios.h
[...]
```

În exemplul de mai sus, se caută recursiv în `/usr/include` fişierele (`-type f`) al căror nume corespunde expresiei regulate `'*term*.h'` şi se afişează acele fişiere (`-print`).

Sintaxa find este, aşadar,

```
find director optiuni_potrivire actiuni
```

unde:

- `director` este directorul în care se face căutarea (recursiv);
- `optiuni_potrivire` sunt opțiuni de căutare în sistemul de fişiere (adâncimea căutării, tipul intrării, nume, deținător, permisiuni, etc.);
- `actiuni` reprezintă acțiunile întreprinse în momentul găsirii unei intrării corespunzătoare; acțiunea implicită este `-print` (de afişare).

Opțiuni de potrivire utile sunt:

- tipul intrării se precizează cu ajutorul opțiunii `-type`; astfel, `-type f` înseamnă un fişier, `-type d` înseamnă un director, `-type l` înseamnă o legătură simbolică, etc.; exemplu: afişarea legăturilor simbolice din ierarhia utilizatorului curent:

```
razvan@anaconda:~$ find /home/razvan/public_html/ -type l
/home/razvan/public_html/pub/books
/home/razvan/public_html/pictures
/home/razvan/public_html/movies
/home/razvan/public_html/games
/home/razvan/public_html/wiki
/home/razvan/public_html/school/uso/cursuri
[...]
```

- numele intrării se precizează cu ajutorul opțiunii `-name` urmată de o expresie regulată descriind numele; opțiunea `-wholename` este folosită pentru potrivire cu numele complet al intrării (cale absolută); exemplu: afișarea directoarelor ce conțin calea `c++`:

```
razvan@anaconda:~$ find /usr/include/ -type d -wholename '*/c++/*'
/usr/include/c++/4.1.2
/usr/include/c++/4.1.2/backward
/usr/include/c++/4.1.2/bits
/usr/include/c++/4.1.2/debug
/usr/include/c++/4.1.2/ext
/usr/include/c++/4.1.2/ext/pb_assoc
[...]
```

- permisiunile se precizează cu ajutorul opțiunii `-perm`; astfel, `-perm 644` va găsi intrările care au drepturi de citire și scriere pentru utilizator și doar de citire pentru grup și pentru ceilalți; exemplu: afișarea executabilelor suidate de sistem:

```
razvan@anaconda:~$ find /bin -perm 4755
/bin/su
/bin/mount
/bin/umount
/bin/ping
/bin/ping6
```

- timpul ultimului acces sau a ultimei modificări cu ajutorul opțiunilor `-amin`, `-atime`, `-mmin`, `-mtime`; `-amin 20` înseamnă că fișierul a fost accesat în urmă cu `20` de minute, `-amin -20` înseamnă că fișierul a fost accesat în urmă cu cel mult `20` de minute, `-amin +20` înseamnă că fișierul a fost accesat în urmă cu cel puțin `20` de minute; exemplu: aflarea fișierelor modificate în urmă cu mai mult de `100` de zile:

```
razvan@anaconda:~$ find /home/razvan/ -type f -mtime +100
```

- spațiul ocupat pe disc cu ajutorul opțiunii `-size`; `-size +10M` va selecta fișierele care ocupă mai mult de `10 MB`; exemplu: aflarea fișierelor care ocupă mai mult de `200 MB`:

```
razvan@anaconda:~$ find /home/razvan/ -size +200M
```

- deținătorul fișierului cu ajutorul opțiunii `-user`;
- grupul fișierului cu ajutorul opțiunii `-group`.

Acțiunile ce pot fi întreprinse cu ajutorul comenzii find sunt:

- afișarea fișierului se face folosind `-print`; această acțiune este implicită;
- afișarea fișierelor folosind nul-terminatorul ca separator folosind `-print0`; opțiunea este utilă în combinație cu argumentul `-0` la `xargs`; exemplu: mutarea fișierelor `.iso` pe o nouă partiție:

```
$ find . -type f -name *.iso -print0 | xargs -0 -I abc mv abc /mnt/hda6
```

- ștergerea fișierului cu ajutorul `-delete`; exemplu: ștergerea fișierelor temporare din ierarhia utilizatorului local:

```
razvan@anaconda:~$ find . -type f -name '*~' -delete
```

- execuția unei comenzi precizate de utilizator cu ajutorul opțiunii `-exec`; șirul `{}` este utilizat pentru înlocuirea intrării găsite în comanda de după exec; comanda trebuie să se termine cu `;` (punct și virgulă); se recomandă ca atât șirul `{}` cât și caracterul ; să fie citate pentru a preveni interpretarea lor ca niște caractere speciale; exemple includ crearea unei copii a fișierelor mai vechi de `1` an din directorul curent:

```
razvan@anaconda:~/junk/scripting$ find . -type f -mtime +365 -exec cp '{}'
/tmp/'{}'.bkup ';'
```

- adăugarea dreptului de execuție pentru toate directoarele:

```
razvan@anaconda:~/junk/scripting$ find . -type d -exec chmod a+x '{}' ';'
```

- afișarea fișierelor C în care este definită funcția `main`:

```
razvan@anaconda:~$ find school/2001-2006_code/ -type f -name '*.c' -exec
grep -l 'int main*' '{}' ';'
```

Acțiuni asupra fișierelor găsite cu find pot fi executate cu ajutorul opțiunii `-exec` sau prin folosirea `xargs`.

11.6.3.1 Exemple

În continuare sunt prezentate câteva exemple utile de lucru cu find:

- editarea fișierelor `.c` din ierarhia curentă care folosesc funcția `printf`:

```
razvan@anaconda:~$ vi $(find school/2001-2006_code/ -type f -name '*.c' -
exec grep -l -w 'printf' '{}' ';')
```

- arhivarea fișierelor mai vechi de `30` de zile:

```
razvan@anaconda:~/junk/scripting$ find . -depth -type f -mtime 30 | tar --
files-from=- --null -czf out.tgz
```

- ștergerea fișierelor utilizatorului local din `/tmp`:

```
razvan@anaconda:~/junk/scripting$ find /tmp -type f -user $(whoami) -
delete
```

*NOTĂ: Informații complete de comenzile de lucru cu fișierele se găsesc în paginile info asociate (**info find**).*

11.7 Expandarea în shell

De-a lungul secțiunilor anterioare s-au prezentat diferite metode de expandare în shell. Expandarea unei variabile la valoarea ei, expandarea unei comenzi, expandare aritmetică. Secțiunea de față oferă o recapitulare structurată a acestor mecanisme.

11.7.1 Simbolul $

Simbolul `$` este folosit pentru expandarea unei variabile, comenzi sau expresii și posibilitatea transmiterii rezultatului acesteia. Caracterul are o semnificație specială, drept pentru care va trebui citat (folosind ghilimele, apostrof, backslash) în cazul în care se dorește folosirea sa literală.

11.7.1.1 Variabile

O variabilă are un nume şi o valoare. Aşa cum s-a specificat, o variabilă nu are tip în shell. Poate fi interpretată ca număr sau ca şir în funcţie de situaţie. Spre exemplu, în nici una din cazurile de test de mai jos nu se generează eroare:

```
razvan@anaconda:~/junk/scripting$ b=3
razvan@anaconda:~/junk/scripting$ test "a" = $b
razvan@anaconda:~/junk/scripting$ test 1 -eq $b
```

Numele variabilei poate fi încadrat între acolade. Acestea sunt opţiunale dar sunt utile pentru a proteja numele variabilei de caracterele următoare. În exemplul de mai jos nu se afişează dacă nu folosim acolade:

```
razvan@anaconda:~/junk/scripting$ echo ${b}a
3a
```

Folosind caracterul `$` se pot face diverite expandări asupra variabilei:

- determinarea numărului de litere compun variabila:

```
razvan@anaconda:~/junk/scripting$ var="alfanumeric"
razvan@anaconda:~/junk/scripting$ echo ${#var}
11
```

- porţiuni din numele variabilei:

```
razvan@anaconda:~/junk/scripting$ echo ${var:3}
anumeric
razvan@anaconda:~/junk/scripting$ echo ${var:3:6}
anumer
```

11.7.1.2 Expandarea unei comenzi

Expandarea unei comenzi înseamnă executarea acesteia şi reţinerea rezultatului comenzii. Acest lucru se poate face în două moduri:

```
$(comanda)
```

sau

```
`comanda`
```

(caracterul folosit este ` - backquote - se află pe tasta cu ~ de lânga tasta `1`)
Rezultatul unei comenzi poate fi reţinut într-o variabilă şi folosit ulterior.

11.7.1.3 Expandarea aritmetică

Expandarea aritmetică presupune evaluarea unei expresii şi reţinerea rezultatului acesteia. Pentru aceasta se foloseşte operatorul `$(( ... ))`:

```
$(( expresie ))
```

Dacă nu este nevoie de reţinerea rezultatului se poate folosi operatorul `(( ... ))`

```
razvan@anaconda:~/junk/scripting$ a=1
razvan@anaconda:~/junk/scripting$ ((a++))
razvan@anaconda:~/junk/scripting$ echo $a
2
```

11.7.2 Expresii regulate în shell

Shell-ul dispune de mecanimse de expresii regulate pentru a permite expandarea numelor de fişiere.

Construcțiile care permit expandarea numelor de fișiere sunt:

- acoladele `{}` permit precizarea unor opțiuni pentru expandare; astfel, expresia `{a,b,c}.d` se va expanda la `a.d b.d c.d`;
- `*` înseamnă expandarea la orice șir de caractere; astfel `*.c` înseamnă toate fișiere care au extensia `.c`, iar `a*.c` reprezintă toate fișierele care încep cu litera a și se termină în `.c`;
- `?` înseamnă potrivirea cu un singur caracter (oricare ar fi acela);
- `[ ... ]` permit specificarea unei clase de caractere; astfel, `[a-z]` înseamnă litere mici `[a-d0-3A-D]` înseamnă literele de la `a` la `d`, de la `A` la `D` sau cifrele de la `0` la `3`; `[^a-z]` înseamnă orice mai puțin literele de la `a` la `z`.

Exemple de utilizare a expresiilor regulate shell sunt prezentate în continuare:

- afișarea tuturor fișierelor C din directorul curent:

```
$ ls *.c
```

- afișarea tuturor fișierelor de tip png, jpg sau gif:

```
$ ls *.{png,jpg,gif}
```

- afișarea tuturor fișierelor al căror nume începe cu 3 cifre:

```
$ ls [0-9][0-9][0-9]*
```

11.8 Parametrii unui script shell

La fel ca orice alt program, un script shell poate primi argumente în linia de comandă. Acestea pot fi folosite pentru a oferi un mod de configurare a scriptului în momentul execuției.

În C modul în care se realizează accesul la argumentele în linia de comandă este prin intermediul variabilelor `argc` și `argv`. Astfel, variabila `argc` indică numărul de argumente primite în linia de comandă (incluzând numele executabilului), iar `argv` este vectorul de argumente: `argv[0]` este numele executabilului, `argv[1]` numele primului argument, `argv[argc-1]` este ultimul argument.

În Bash, accesul la argumente se realizează prin intermediul variabilei `$#`, reprezentând numărul de argumente, și al variabilelor `$1`, `$2`, etc. reprezentând argumentele efective. Ca exemplu, vom folosi scriptul de mai jos, care nu face altceva decât să afișeze argumentele primite:

```
razvan@anaconda:~/junk/scripting$ cat -n args.bash
     1  #!/bin/bash
     2
     3  echo "Scriptul are $# argumente."
     4  echo "Numele scriptului este $0."
     5  echo "Argumentele scriptului sunt '$@'."
     6
     7  i=1
     8
     9  while test $i -le $#; do
    10          echo "Argumentul $i este ${!i}."
    11          ((i++))
    12  done
razvan@anaconda:~/junk/scripting$ bash args.bash alfa beta gamma delta
Scriptul are 4 argumente.
Numele scriptului este args.bash.
Argumentele scriptului sunt 'alfa beta gamma delta'.
Argumentul 1 este alfa.
Argumentul 2 este beta.
Argumentul 3 este gamma.
Argumentul 4 este delta.
```

În cazul exemplului de mai sus, se trimit 4 argumente. Dacă variabila argc numără inclusiv executabilul, variabila `$#` contorizează doar argumentele. Variabila `$0` conține numele executabilului. Variabila `$@` conține toate argumentele separate prin spațiu (blank) (linia 5). Folosind variabila `i` se parcurg toate argumentele primite în linia de comandă (linia 9) şi se afişează (linia 10). Se observă că am folosit construcția `${!i}`, pentru a expanda variabila `i` şi apoi variabila care reprezintă argumentul primit din linia de comandă.

Un al mod de a realiza parcurgerea argumentelor primite în linia de comandă ar fi fost următorul:

```
i=1
for arg in $@; do
        echo "Argumentul $i este $arg."
        ((i++))
done
```

Variabilele `$1`, `$2`, ..., `${N}`, reprezentând argumentele primite din linia de comandă, poartă numele de **parametri poziționali**.

11.8.1 Comanda shift

Comanda `shift`, după cum îi spune numele, este folosită la deplasarea argumentelor din linia de comandă în parametri poziționali. În exemplul de mai jos, se face parcurgerea parametrilor poziționali folosind comanda `shift`:

```
i=1
while ! test -z $1 ; do
        echo "Argumentul $i este $1."
        ((i++))
        shift
done
```

Comanda `shift` este responsabilă cu deplasarea spre stânga a argumentelor unui script shell.Astfel, după rularea comenzii `shift`, se pierde informație despre primul argument, variabila `$#` scade cu 1, variabila `$1` reprezintă al doilea argument, variabila `$2` reprezintă al treilea argument, etc.

Comanda `shift` poate primi un argument reprezentând contorul de deplasare. Implicit (fără argument) acesta este 1.

11.8.2 Parametri speciali

După cum s-a observat există un set de variabile speciale folosite de shell pentru prelucrarea argumentelor în linia de comandă:

- `$#` reprezintă numărul de argumente din linia de comandă;
- `$0` reprezintă numele scriptului shell;
- `$1`, `$2`, ..., `${N}` reprezintă argumentele din linia de comandă;
- `$@` reprezintă lista de argumente separate prin spații.
- În afara acestor variabile mai există câțiva parametri speciali, prezentați în continuare:
- `$?` reprezintă valoarea de retur a ultimei comenzi;
- `$$` reprezintă pid-ul procesului bash curent;
- `$!` reprezintă pid-ul ultimului proces lansat în fundal.

11.8.3 Un exemplu

Un exemplu util este un script shell care să uşureze lucrul cu comanda find în momentul căutării unor fişiere după nume. Scriptul este prezentat în continuare:

```
razvan@anaconda:~/junk/scripting$ cat -n myfind.bash
     1  #!/bin/bash
     2
     3  if test $# -ne 2; then
     4          echo "Two arguments are required."
     5          exit 1
     6  fi
     7
     8  if ! test -d $1; then
     9          echo "First argument is not a directory."
    10          exit 1
    11  fi
    12
    13  find "$1" -type f -name "$2"
razvan@anaconda:~/junk/scripting$ bash myfind.bash . "*.bash"
./while10.bash
./hw.bash
./for_ls.bash
./service_ip.bash
./while_cat.bash
./rss.bash
[...]
```

Se observă că se transmit ca argumente directorul în care se face căutarea şi expresia regulată asociată numelui fişierului. În afara apelării `find` (linia 13), se face verificarea numărului de argumente (liniile 3-6) şi dacă primul argument este un director (liniile 8-11).

11.9 Funcții

La fel ca majoritatea limbajelor de programare, shell scriptingul permite folosirea de funcții. În general funcțiile sunt bucăți de cod folosite pentru a separa o funcționalitate sau pentru refolosire (nu se scrie de două ori acelaşi cod). Au, de asemenea, un rol important în mărirea lizibilității codului.

În shell scripting, funcțiile intervin în cazul scripturilor de dimensiune mai mare, unde există bucăți de cod care sunt folosite de mai multe ori.

11.9.1 Sintaxa unei funcții

Prezentăm sintaxa comenzii prin următorul exemplu, în care se poate șterge sau adăuga un utilizator în sistem:

```
anaconda:/home/razvan/junk/scripting# cat -n user.bash
     1  #!/bin/bash
     2
     3  if test $# -ne 1; then
     4          echo "One argument required: a or d (add or delete)."
     5          exit 1
     6  fi
     7
     8  username=testuser
     9  groupname=testgroup
    10
    11  function add ()
    12  {
    13          cat /etc/group | grep $groupname &> /dev/null
    14          if test $? -eq 1; then          # no group
    15                  addgroup $groupname
    16                  echo "Group $groupname didn't exist so it was created."
    17          fi
    18          useradd -m -d /home/$username -g $groupname -s /bin/bash $username
    19          echo "User $username added. Use passwd to change password."
    20  }
    21
    22  function delete ()
    23  {
    24          userdel -r $username
    25          echo "User $username deleted."
    26  }
    27
    28  case $1 in
    29          "a") add;;
    30          "d") delete;;
    31          *) echo "Required arguments are 'a' or 'd'."; exit 1;;
    32  esac
anaconda:/home/razvan/junk/scripting# bash user.bash a
Adding group `testgroup' (GID 1031) ...
Done.
Group testgroup didn't exist so it was created.
User testuser added. Use passwd to change password.
anaconda:/home/razvan/junk/scripting# bash user.bash d
User testuser deleted.
```

În exemplul de mai sus funcțiile sunt `add` și `delete` și sunt folosite, respectiv, pentru a adăuga sau elimina un utilizator în/din sistem. Sintaxa de definiție a unei funcții este, așadar:

```
function nume_functie ()
{
    cod shell
}
```

Se observă că o funcție shell are un antet și un corp. În antet se prezintă numele funcției (`nume_functie`), iar în corp se regăsesc comenzile executate în momentul apelării funcției. Cuvântul `function` este cuvânt rezervat și poate fi omis. Astfel o funcție definită:

```
function testare ()
{
    ...
}
```

poate fi definită și astfel:

```
testare ()
{
    ...
}
```

Pentru a apela o funcție se folosește numele funcție, exact ca un exemplu de apel de comandă.

11.9.2 Parametrii unei funcții

În mod evident, o funcție poate primi și argumente. Modul de definire a unei funcții nu se schimbă; altfel spus, în shell scripting o funcție nu are parametri formali. Totuși, modul de apel se schimbă prin precizarea parametrilor pentru funcție.

Pentru a exemplifica modul în care se transmit parametrii unei funcții vom folosi un exemplu de calcul a sumei unor numere naturale transmise ca argumente funcției `sum_func`:

```
razvan@anaconda:~/junk/scripting$ cat -n sum_func.bash
     1  #!/bin/bash
     2
     3  function sum_func ()
     4  {
     5          sum=0
     6          while ! test -z $1; do
     7                  sum=$(($sum + $1))
     8                  shift
     9          done
    10  }
    11
    12  sum_func 1 2 3 4 5
    13  echo "Sum: $sum"
    14
    15  sum_func 2 3 5 7 11 13 17 19
    16  echo "Sum: $sum"
razvan@anaconda:~/junk/scripting$ bash sum_func.bash
Sum: 15
Sum: 77
```

Se observă că argumentele unei funcții se obțin tot folosind parametrii poziționali (`$1`, `$2`, etc.). Variabila `$#` reprezintă numărul de argumente transmise funcției. Variabila `$0` rămâne neschimbată și reprezintă numele scriptului. Se poate, de asemenea, folosi comanda `shift`. În momentul în care funcția se întoarce, valorile parametrilor poziționali sunt restaurate. Numele funcției este reținut în variabila `FUNCNAME`.

Variabila `sum` a fost definită în cadrul funcției și este vizibilă și în exterior. Dacă se dorește ca o variabilă să fie vizibilă numai în interiorul unei funcții aceasta trebuie prefixată folosind cuvântul cheie `local`.

11.9.3 Scripturile de pornire Bash

Scripturile de pornire Bash sunt un set de fișiere care sunt intepretate de Bash la pornirea unei sesiuni shell. Acestea se ocupă de actualizarea variabilelor de mediu și de rularea unor comenzi personalizate de utilizator; un exemplu îl constituie configurarea de alias-uri.

Sesiunile de shell pot fi clasificate în:

- sesiuni interactive/non-interactive după cum permit sau nu rularea de comenzi prin prezentarea unui prompt utilizatorului; o sesiune non-interactivă este, de obicei, folosită pentru rularea unui script shell

- sesiuni de autentificare (login)/non-autentificare (non-login); sesiunile de emulator de terminal (`gnome-terminal`, `konsole`, `xterm`) sunt sesiuni non-login
- Fişierele de pornire sunt interpretate de sesiunile de shell interactive în modul următor:
- o sesiune de shell login va interpreta, în ordine, `/etc/profile`, `~/.bash_profile`, `~/.bash_login` şi `~/.profile`; la încheierea sesiunii se interpretează fişierul `~/.bash_logout`
- o sesiune de shell non-login va interpreta comenzile din `~/.bashrc`

Fişierul `~/.bashrc` este fişierul recomandat pentru personalizarea sesiunilor de shell ale unui utilizator. Întrucât un shell login nu interpretează `~/.bashrc`, fişierul `~/.bash_profile` va conţine, de obicei, o linie de forma:

```
# include .bashrc if it exists
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```

11.9.4 Variabile de mediu

După cum am specificat, unul din rolurile importante ale scripturilor de pornire shell este configurarea variabilelor de mediu.

Variabilele de mediu sunt un set de variabile care afectează modul în care un proces rulează. În mod obişnuit, în sistemele Unix, un proces moşteneşte variabilele de mediu ale părintelui său. Astfel, pentru procesul asociat execuţiei unei comenzi, variabilele de mediu sunt cele ale shell-ului.

Variabilele de mediu sunt, de obicei, configurate în scripturile de pornire Bash. Declaraţia unei variabile de mediu se face în formatul

```
export VARIABLE=value
```

Dacă nu se precizează `export`, atunci variabila nu va fi moştenită de procesele fiu. În mod obişnuit, variabilele de mediu sunt denumite cu majuscule.

Vizualizarea variabilelor de mediu existente la un moment dat se realizează cu ajutorul comenzii `env` sau `set`:

```
razvan@anaconda:~/junk/scripting$ env
SSH_AGENT_PID=7985
TERM=xterm
SHELL=/bin/bash
 [...]
```

Exemple de variabile de mediu sunt:

`PATH`: reţine o listă de directoare folosite de shell pentru a găsi executabilele asociate comenzii; lista este separată prin : (două puncte); dacă se doreşte adăugarea unui director `~/bin` la variabila de mediu, se foloseşte comanda:

```
razvan@anaconda:~/junk/scripting$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
razvan@anaconda:~/junk/scripting$ export PATH="$PATH":~/bin
razvan@anaconda:~/junk/scripting$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:/home/razvan/bin
```

- `HOME` reţine directorul de bază al utilizatorului curent
- `USER` este numele utilizatorului curent
- `EDITOR` este editorul implicit

11.10 Studii de caz

11.10.1 Schimbarea promptului shell

Promptul este şirul de caractere oferit de shell unde utilizatorul poate introduce comenzi. În Bash, un prompt care se termină în caracterul `$` înseamnă o sesiune de shell de utilizator neprivilegiat; un prompt care se termină în caracterul `#` este asociat unei sesiuni de shell de root.

Aspectul promptului este controlat prin intermediul unei variabile speciale, anume `PS1`. Aceasta controlează modul în care promptul este oferit utilizatorului. Utilizarea comenzii echo permite inspectarea variabilei:

```
razvan@anaconda:~$ echo $PS1
\u@\h:\w\$
```

Se observă că variabila cuprinde o serie de caractere speciale citate cu ajutorul backslash:

- `\u` înseamnă numele utilizatorului; în exemplul de mai sus se traduce în şirul `razvan`
- `\h` înseamnă numele sistemului
- `\w` înseamnă afişarea directorului curent, cu abrevierea directorului de bază la caracterul `~`
- `\$` înseamnă `#` pentru root sau `$` în rest

Promptul poate fi schimbat la un şir de caractere simplu:

```
razvan@anaconda:~$ PS1="prompt: "
prompt:
```

Poate fi, însă, personalizat folosind o serie de alte caractere speciale ca în exemplele de mai jos:

```
prompt: echo PS1="\u-\h[\d]\$ "
PS1=\u-\h[\d]$
prompt: PS1="\u-\h[\d]\$ "
razvan-anaconda[Fri Sep 14]$ PS1="\u+\h \W (\A)"
razvan+anaconda ~ (18:08)cd junk/
razvan+anaconda junk (18:08)cd ~/junk/conv/
razvan+anaconda conv (18:09)
```

Secvenţe speciale care au fost folosite în exemplele de mai sus sunt:

- `\d` înseamnă data curentă
- `\A` înseamnă timpul curent
- `\W` înseamnă directorul curent în cale relativă (`\w` afişează calea completă)

O listă completă cu secvenţele speciale de personalizare a promptului găsiţi la http://tldp.org/HOWTO/Bash-Prompt-HOWTO/bash-prompt-escape-sequences.html.

Variabila PS1 este definită în `/etc/profile` şi `~/.bashrc`. Pentru configurarea permanentă a acesteia se recomandă utilizarea fişierului `~/.bashrc`

11.10.1.1 Folosirea de culori în prompt

Promptul poate avea culoare prin folosirea secvenţei speciale `\e[`. Astfel, dacă se doreşte afişarea unui prompt de culoare roşie se foloseşte comanda:

```
razvan@anaconda:~/junk/conv$ PS1="\e[0;31m\u@\h:\w\$ \e[m"
```

Pentru folosirea unui prompt de culoare verde comanda este:

```
razvan@anaconda:~/junk/conv$ PS1="\e[0;32m\u@\h:\w\$ \e[m"
```

Secvențele speciale `\e[x;ym` și `\e[m` definesc spațiul de aplicabilitate al unei culori. Secvența `x;y` definește culoarea utilizată. Secvențe posibile sunt:

- `0;30` – negru
- `0;31` – roșu
- `0;32` – verde
- `0;33` – galben
- `0;34` – albastru
- `0;35` – magenta
- `0;36` – cyan
- `0;37` – alb

În lista de mai sus `x` are valoare `0`. Se poate folosi valoarea `1` pentru o versiunea deschisă a culorii. În mod evident, se poate asocia câte o culoare numai unei anumite componente a promptului ca în exemplul de mai jos:

```
razvan@anaconda:~$ PS1="\e[0;36m\u\e[m@\e[0;34m\h\e[m:\e[0;31m\w\e[m\$ "
```

În exemplul de mai sus, în cadrul promptului, numele utilizatorului va fi afișat folosind cyan, numele sistemului folosind albastru, iar numele directorului curent folosind roșu.

11.10.2 Batch scripting în Windows

În Windows, scripturile shell au ca echivalent scripturi batch. Un script batch are de obicei extensia `.bat`. Programul folosit pentru rularea scripturilor batch este cmd (interpretorul de comenzi pe Windows).

La fel ca un script shell un script batch permite execuția de comenzi native Windows și programarea cu ajutorul variabilelor, a instrucțiunilor de decizie și a instrucțiunilor de ciclare

11.10.2.1 Comenzi Windows

Rularea unei comenzi în Windows se realizează prin intermediul interpretorului de comenzi Windows. Pentru lansarea acestuia, trebuie lansat, în Windows XP, `Command Prompt`: `Start->All Programs->Accesories->Command Prompt`.

În afara comenzilor obișnuite pentru lucrul cu sistemul de fișiere (`dir`, `cd`, `mkdir`, `rmdir`, `del`, `copy`, `move`, etc.), există și alte comenzi utile. Lista cu toate comenzile Windows se poate obține folosind comanda `help`:

```
C:\Documents and Settings\Razvan.RAGNAROK>help
For more information on a specific command, type HELP command-name
ASSOC    Displays or modifies file extension associations.
AT       Schedules commands and programs to run on a computer.
ATTRIB   Displays or changes file attributes.
BREAK    Sets or clears extended CTRL+C checking.
CACLS    Displays or modifies access control lists (ACLs) of files.
[...]
```

În tabelul de mai jos este prezentat un tabel cu comenzile utile Linux și echivalentul lor în Windows:

Linux	Windows	Efect
`pwd`	`cd`	afișează directorul curent
`ls`	`dir`	listează conținut director
`cd`	`cd`	schimbă director
`mkdir`	`mkdir, md`	creează director
`rmdir`	`rmdir, rm`	șterge director

touch	-	creează fişier
rm	del, erase	şterge fişier
cp	copy	copiază fişier
mv	move	mută fişier
cat	type	afişează conţinut fişier
echo	echo	afişează mesaj
less, more	more	afişare paginată fişier
cmp	comp	compară fişiere
diff	fc	compară şi afişează diferenţe
file	ftype	afişează tipul fişierului
grep	findstr	caută în fişier
shift	shift	schimbă parametrii poziţionali
sort	sort	sortează intrarea

Tabelul 11.2 Comenzi utile Linux şi Windows

11.10.2.2 Operatori

Interpretorul de comenzi Windows are, de asemenea, operatori de redirectare sau de comunicare între procese. În exemplul de mai jos, dacă dorim să reţinem în fişierul del.txt informaţii despre comenzile care se ocupă cu ştergerea de informaţii vom folosi comanda:

```
  C:\Documents and Settings\Razvan.RAGNAROK>help | findstr "[Dd]elete" >
del.txt
  C:\Documents and Settings\Razvan.RAGNAROK>type del.txt
  DEL      Deletes one or more files.
  ERASE    Deletes one or more files.
  LABEL    Creates, changes, or deletes the volume label of a disk.
  [...]
```

11.10.2.3 Variabile

Ca şi în Bash, variabilele nu au un tip. Definirea acestora se realizează cu ajutorul comenzii `set`:

```
  C:\Documents and Settings\Razvan.RAGNAROK>set a=3
  C:\Documents and Settings\Razvan.RAGNAROK>echo %a%
  3
```

Afişarea conţinutul unei variabile nu se mai realizează prefixând-o cu ajutorul caracterului `$` ci folosind caracterul `%` înainte şi după numele variabilei.

Ca şi în Bash, comanda `set` este folosită şi pentru afişarea variabilelor curente:

```
  C:\Documents and Settings\Razvan.RAGNAROK>set
  a=3
  ALLUSERSPROFILE=C:\Documents and Settings\All Users
  APPDATA=C:\Documents and Settings\Razvan.RAGNAROK\Application Data
  CommonProgramFiles=C:\Program Files\Common Files
  COMPUTERNAME=RAGNAROK
  ComSpec=C:\WINDOWS\system32\cmd.exe
  FP_NO_HOST_CHECK=NO
  HOMEDRIVE=C:
  HOMEPATH=\Documents and Settings\Razvan.RAGNAROK
  [...]
```

Variabile de mediu

Dacă în Bash variabilele de mediu sunt definite în fişierele de pornire, în Windows trebuie folosit un meniu special de configurare. Pentru accesul acestuia se foloseşte click dreapta pe `My Computer->Properties->Advanced->Environment Variables`.

Variabila `PATH` deţine, ca şi în cazul Bash, lista de directoare ale căror executabile pot fi rulate din linia de comandă. Spre deosebire de Bash, însă, separatorul folosit este `;` (punct şi virgulă).

În exemplul de mai jos se adaugă directorul `D:\mybin` la calea de executabile:

```
C:\Documents and Settings\Razvan.RAGNAROK>echo %PATH%
c:\program files\imagemagick-6.3.4-q16;C:\Program Files\MiKTeX
2.6\miktex\bin;C:
\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;D:\Software\swftools
C:\Documents and Settings\Razvan.RAGNAROK>set PATH=%PATH%;D:\mybin\
C:\Documents and Settings\Razvan.RAGNAROK>echo %PATH%
c:\program files\imagemagick-6.3.4-q16;C:\Program Files\MiKTeX
2.6\miktex\bin;C:
\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;D:\Software\swftools
;D:\mybin\
```

11.10.2.4 Programarea batch

După cum s-a precizat, echivalentul scripturilor shell din Linux sunt scripturile batch din Windows. Acestea au, de obicei, extensia `.bat`. Exemplul cel mai simplu de script batch este prezentat mai jos:

```
C:\DOCUME~1\RAZVAN~1.RAG>type hw.bat
@echo off
echo "Hello, World!"
C:\DOCUME~1\RAZVAN~1.RAG>hw.bat
"Hello, World!"
```

În mod implicit, rularea unui script batch duce la afişarea comenzilor întâlnite (echivalent cu `set -x` din Bash). Pentru a preîntâmpina acest lucru se foloseşte directiva `@echo off`.

Un emulator al comenzii `dir` foloseşte `for` şi este prezentat în continuare:

```
C:\DOCUME~1\RAZVAN~1.RAG>type dir.bat
@echo off
for %%f in (*) do echo %%f

C:\DOCUME~1\RAZVAN~1.RAG>dir.bat
.gtk-bookmarks
del.txt
dir.bat
gsview32.ini
hw.bat
logfile.txt
PUTTY.RND
winscp.RND
_viminfo
```

Instrucţiunea if are o sintaxa asemănătoare instrucţiunii echivalente din Bash:

```
C:\DOCUME~1\RAZVAN~1.RAG>set a=3
C:\DOCUME~1\RAZVAN~1.RAG>if %a% EQU 3 echo "true"
"true"
```

Mai multe despre comenzile instrucţiunile `if` şi `for` se pot afla prin consultarea informaţiei de ajutor: `if /?` şi `for /?`

11.10.3 PowerShell pe Windows

PowerShell este un nou interpretor de comenzi dezvoltat de Microsoft. Se bazează pe programare orientată obiect şi pe versiunea 2.0 a platformei de dezvoltare .NET. Windows PowerShell este gratis; poate fi descărcat de pe site-ul asociat:

http://www.microsoft.com/windowsserver2003/technologies/management/powershell/default.mspx.

Windows PowerShell urmează, într-o oarecare măsură, ideea din spatele utilitarelor shell Unix, anume de a realiza sarcini complexe prin combinarea unor componente deja existente. Aceste componente sunt denumite **cmdlets** şi sunt, de fapt, instanţe ale unor clase .NET. Deosebirea de shellurile Unix este faptul că în loc de a crea o bandă de transmitere a informaţiei în format text (pipeline), cmdlet-ii comunică folosind obiecte.

11.10.3.1 Cmdlets

Cmdlet-ii reprezintă blocurile de bază pentru PowerShell echivalent cu comenzile de bază Unix folosite de Bash.

Astfel, pentru listarea proceselor din sistem se foloseşte cmdlet-ul `get-process`. Acesta prezintă aliasul `ps` şi este echivalent comenzii `ps` din Linux. Pentru listarea conţinutului unui director se foloseşte cmdlet-ul get-childitem cu aliasul `ls`.

Dacă, spre exemplu, un utilizator doreşte omorârea proceselor al căror nume începe cu litera `p`, se foloseşte comanda:

```
PS> get-process p* | stop-process
```

Folosind concepte din programarea orientată obiect, se pot extrage câmpuri din rezultatul unei comenzi. Astfel, dacă se doreşte aflarea timpului ultimului acces pentru un directorul `Cookies`, se foloseşte comanda

```
PS> $(get-item Cookies).lastaccesstime
```

Pentru afişarea proceselor din sistem în formă tabelară se poate folosi comanda

```
PS> Get-Process | Format-Table Id, Name
  Id Name
  -- ----
2876 alg
 532 ApntEx
2044 Apoint
3448 calc
1824 CFSvcs
```

11.10.3.2 Variabile şi structuri de control

PowerShell permite crearea de scripturi. Ca şi Bash, printre facilităţile precizate se numără completare automată, utilizarea de variabile, structuri de control şi funcţii.

Spre deosebire de Bash, variabilele PowerShell au un tip:

```
PS C:\> $b = 3.1415926
PS C:\> $b
3.1415926
PS C:\> $b.GetType().Name
```

Echivalentul for din Bash este, în Windows PowerShell, `foreach`:

```
PS C:\WINDOWS\system32> 1..10 | foreach { $_ * 2 }
2
4
6
8
10
12
14
16
18
20
```

NOTĂ: *Informații complete despre Windows PowerShell se găsesc în documentația care soseşte cu pachetul de instalare.*

Cuvinte cheie

- shell
- bash
- script shell
- readline
- editarea comenzilor
- istoricul comenzilor
- reverse search
- autocompletion
- comenzi interne (built-in)
- comenzi externe
- shebang
- `echo`
- `printf`
- `set`
- `exit`
- `read`
- redirectarea intrării/ieşirii/erorii standard
- operatorii `<`, `>`, `2>`, `&>`, `>>`
- here document, here string; operatorii `<<`, `<<<`
- operatorii de secvențiere `;`, `&&`, `||`
- operatorul de comunicație `|`
- variabile shell
- caractere speciale shell: `$`, `#`, `*`, `"`, `'` , `\`, `(`, `)`, `{`, `}`, `&`, `|`, `;`, `[`, `]`, `?`, `blank`
- `if`, `test`, comanda de test `[ ... ]`
- `for`, `while`, `until`
- filtre de text: `cat`, `tac`, `nl`, `sort`, `uniq`, `head`, `tail`, `cut`, `tr`, `grep`
- expresie regulată (pattern matching)
- `xargs`, `locate`, `find`
- expandarea shell: expandarea variabilei, substituția comenzii, expandare aritmetică
- parametrii unui script shell; expresiile `$0`, `$n`, `$#`, `$@`, `$?`, `$$`, `$!`
- funcții shell
- scripturi de inițializare: `.bashrc`, `.bash_profile`, `/etc/bash_profile`
- variabile de mediu
- batch scripting
- power shell; cmdlet

Întrebări

Care din următoarele nu este un intepretor de comenzi?

- Bash
- PowerShell
- Command Prompt
- Flash

Ce comandă este folosită pentru căutarea unei expresii regulate într-un fişier?

- `tr`
- `grep`
- `cut`
- `sort`

Care din următorii operatori este asociat cu un here document?

- `>`
- `|`
- `&>`
- `<<`

Care din următoarele opțiuni testează egalitatea între două şiruri?

- `set $a == $b`
- `[$a == $b]`
- `test $a == $b`
- `{ $a == $b }`

Care este efectul comenzii sort –n file.txt?

- Sortează liniile nevide din fişier
- Sortează liniile vide din fişier
- Sortează liniile unice din fişier
- Sortează liniile în format numeric

Dacă un utilizator doreşte substituția caracterului a cu A, ce NU îl va ajuta?

- `vim`
- `cut`
- `sed`
- `tr`

Care din următoarele NU este un avantaj al unui shell script?

- Viteza mare de execuție
- Automatizarea sarcinilor
- Folosirea de componente existente
- Depanare facilă

Care este utilitatea shebang (#!)?

- Permite editarea unui script shell
- Permite asocierea de drepturi de execuție unui script shell
- Prezintă calea către utilitarul folosit pentru rularea scriptului
- Este o linie de comentariu specială într-un script shell

Care din următoarele nu este o instrucțiune de control în Bash?

- `for`
- `while`
- `until`
- `repeat`

Care din următoarele nu afişează intrările din directorul curent?

- `ls`
- `for i in *; do echo $i; done`
- `find . -type f -maxdepth 1`
- `ls *.*`

12 Mediul grafic

A bus station is where a bus stops. A train station is where a train stops.
On my desk I have a workstation...

Ce se învață din acest capitol?

- Prezentarea interfaței grafice în Linux: X Window System
- Componentele X Window System: window manager, desktop environment, display manager
- Comenzile de interacțiune cu X Window System
- Configurarea X Window System: /etc/X11/xorg.conf
- Configurarea sistemului folosind KDE
- Comparație aplicații grafice KDE vs. GNOME
- Configurare Compiz Fussion
- Configurare X peste SSH
- Configurare VNC; configurare RDC

12.1 Interfața grafică în Linux. Componente

Dintre cele trei metode de interacțiune cu sistemul de operare (CLI, TUI, GUI), cel mai utilizat în momentul de fața este GUI. Principalul său avantaj constă în ușurința cu care utilizatorii învață să îl folosească.

Evoluția interfețelor grafice s-a realizat în paralel cu evoluția hardware-ului. Prima încercare de realizare a unei interfețe grafice a avut loc în anii '60 la Institutul de Cercetări al Universității Standford. La acel moment interfața grafică cuprindea un mouse elementar cu ajutorul căruia se puteau accesa link-uri în documentele text (*text hyperlinks*).

În momentul de față interfețele grafice încep să pășească în era 3D. Aero din Windows Vista, Compiz din Linux şi Quartz Extreme din MacOS X oferă utilizatorului o experiență 3D la a cărei realizare participă în mare măsură placa video şi nu procesorul ca până acum.

12.1.1 X Window System

X Window System (cunoscut ca X11 sau ca X) este un cadru (framework) ce permite desenarea de ferestre pe un afişaj de tip raster.

Un afişaj de tip **raster** este un afişaj în care imaginea este reținută ca o matrice de puncte (*bitmap*). De exemplu fotografiile realizate cu o cameră foto digitală sunt imagini raster. Este formatul de imagini cel mai des folosit, având diverse implementări: bmp, jpeg, gif, png, etc. Dezavantajul său constă în faptul că, atunci când imaginea este redimensionată, conținutul noii imagini trebuie aproximat din conținutul imaginii inițiale. Această aproximare se poate realiza

cu rezultate bune în cazul în care imaginea este micşorată. În cazul în care imagine este mărită, rezultatele sunt din ce în ce mai slabe cu cât factorul de scalare este mai mare.

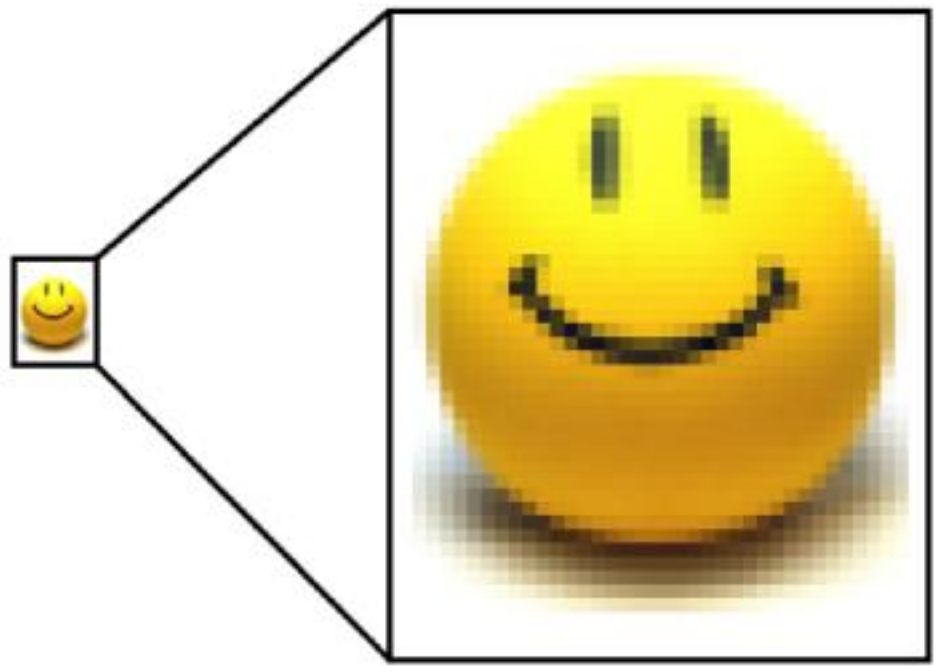


Figura 12-1 Mărirea unei imagini în format raster

Opus acestui tip de afişaj este afişajul **vectorial** în care imaginea este reţinută sub formă elemente geometrice de bază: linii, elipse, puncte şi gradienţi de culoare. Avantajul major al acestor tipuri de imagini este acela că sunt foarte uşor de redimensionat (practic culoarea fiecărui punct din imagine este calculată exact pe baza elementelor ce compun imaginea). În urma redimensionării (micşorare sau mărire) calitatea imaginii nu se pierde. Cu excepţia SVG formatele de imagini vectoriale sunt specifice fiecărui program de prelucrare grafică.

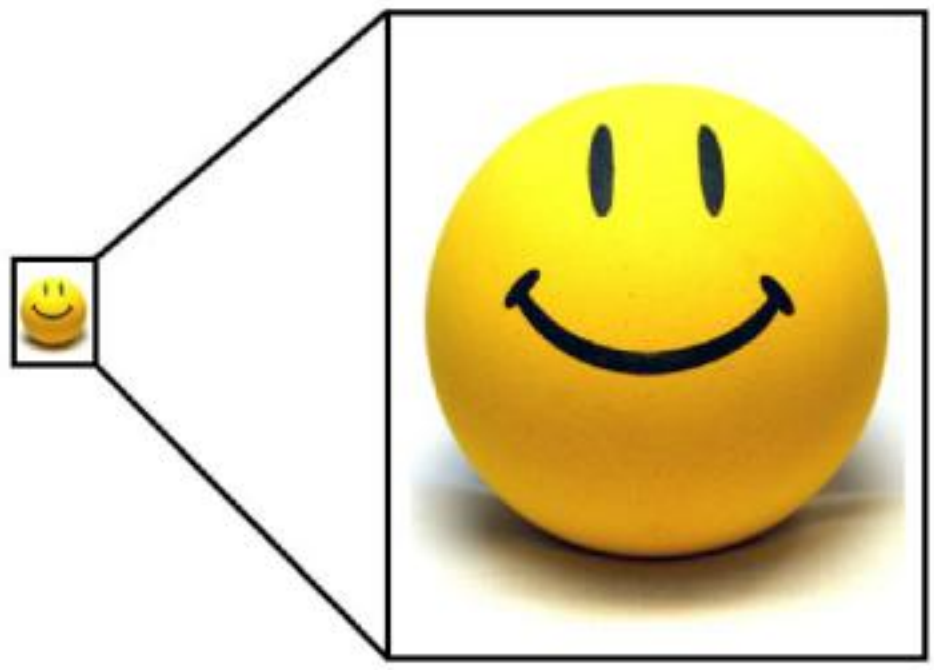


Figura 12-2 Mărirea unei imagini în format vectorial

Monitoarele (CRT sau LCD) afişează imagini în format raster. De aceea imaginea finală care este trimisă către acestea este în format raster.

X Window System oferă un set de utilitare pentru dezvoltarea de interfeţe grafice şi este folosit pentru crearea de interfeţe în Unix, sisteme de operare derivate din Unix şi OpenVMS.

X oferă un cadru de bază pentru crearea unor interfețe grafice: desenarea şi mutarea ferestrelor pe ecran, interacțiunea cu mouse-ul şi interacțiunea cu tastatura. Restul este lăsat în seama programelor, motiv pentru care aspectul vizual al interfețelor grafice bazate pe X diferă destul de mult de la un sistem de operare la altul, sau chiar între versiunile aceluiaşi sistem de operare.

X a fost creat în 1984 la MIT. Versiunea curenta a protocolului este X11 şi a apărut în 1987. În momentul de fata proiectul X este condus de către X.Org Foundation şi a ajuns la versiunea X11 Release 7.2.

12.1.2 Arhitectura X Window System

X Window System are o arhitectură de tip client-server: un sever X comunică cu mai multe programe client. Serverul acceptă cereri de desenare a obiectelor grafice şi trimite înapoi către clienți intrările (*inputul*) de la utilizator (preluate de la tastatura, mouse, etc).

Serverul X este responsabil de administrarea resurselor ce permit interacțiunea cu utilizatorul. Aceste resurse sunt: ecranul, tastatura, mouse-ul, touchpad-ul, stylus-ul, joystick-ul, etc. Serverul X rulează pe calculatorul care interacționează cu utilizatorul şi trimite intrările (*inputul*) primit de la acesta către clienți. Clienții procesează informația primită de la server şi îi trimit acestuia înapoi informații pe care serverul le va afişa utilizatorului pe ecran. Exemple de servere: XFree86, X.Org.

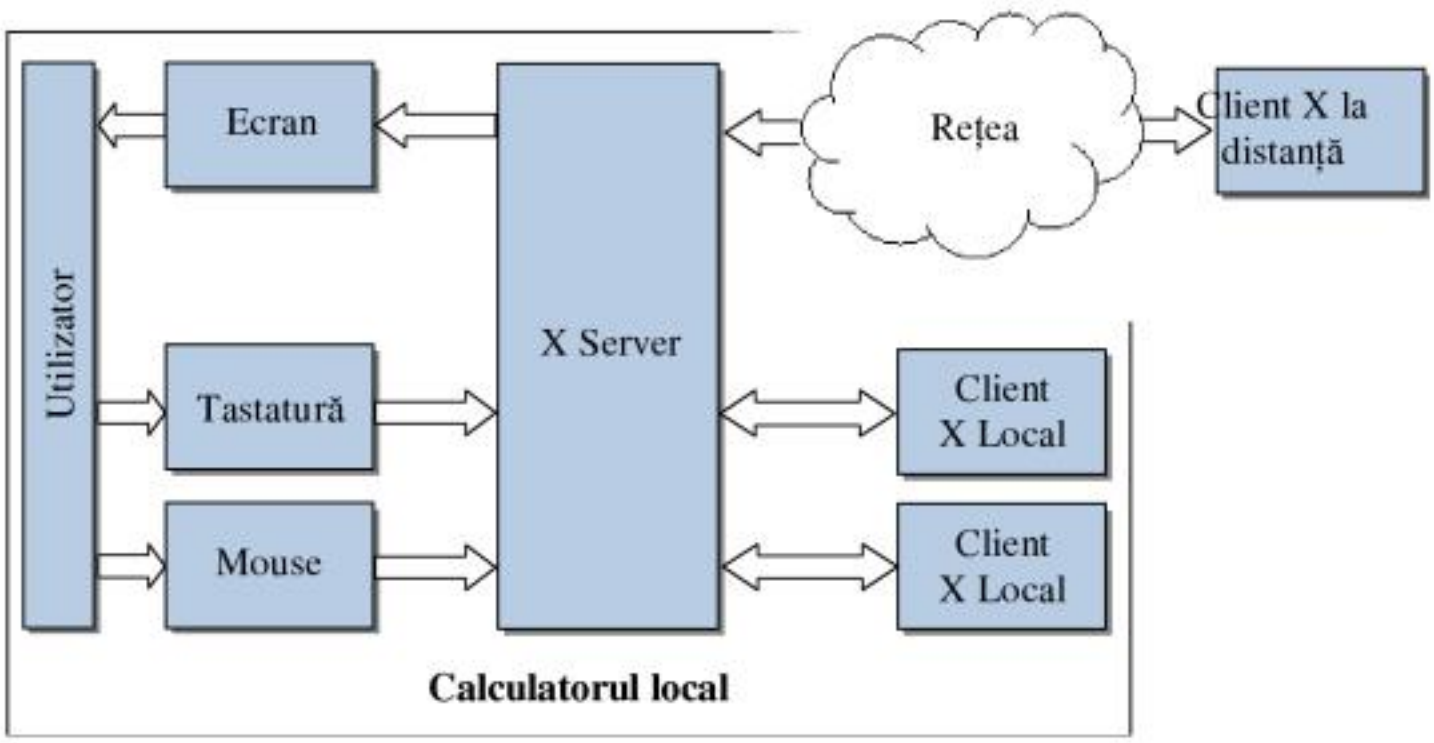


Figura 12-3 Arhitectura X Window System

Clienții X se conectează la un server X pentru a folosi resursele administrate de acesta. Ei cer serverului să afişeze pe ecran obiecte grafice şi preiau de la acesta intrările (*inputul*) utilizatorului. Clienții pot rula local (pe acelaşi calculator cu serverul) sau pot rula pe un alt calculator aflat la distanță. Exemple de clienți X: un browser web, un client de e-mail, un program de mesagerie instant.

Protocolul X este protocolul X utilizat în comunicația dintre clienți şi server.

Comunicația dintre clienți şi server se desfăşoară un pic diferit față de modelul general al comunicației client-server. În general clientul, ca localitate, se află pe calculatorul local iar serverul se află pe un calculator aflat la distanță. În cazul *X Window System* lucrurile stau invers. Serverul se află pe calculatorul local pentru a putea trimite şi primi informații de la utilizator. Clienți se pot afla tot pe calculatorul local sau pe unul aflat la distanță.

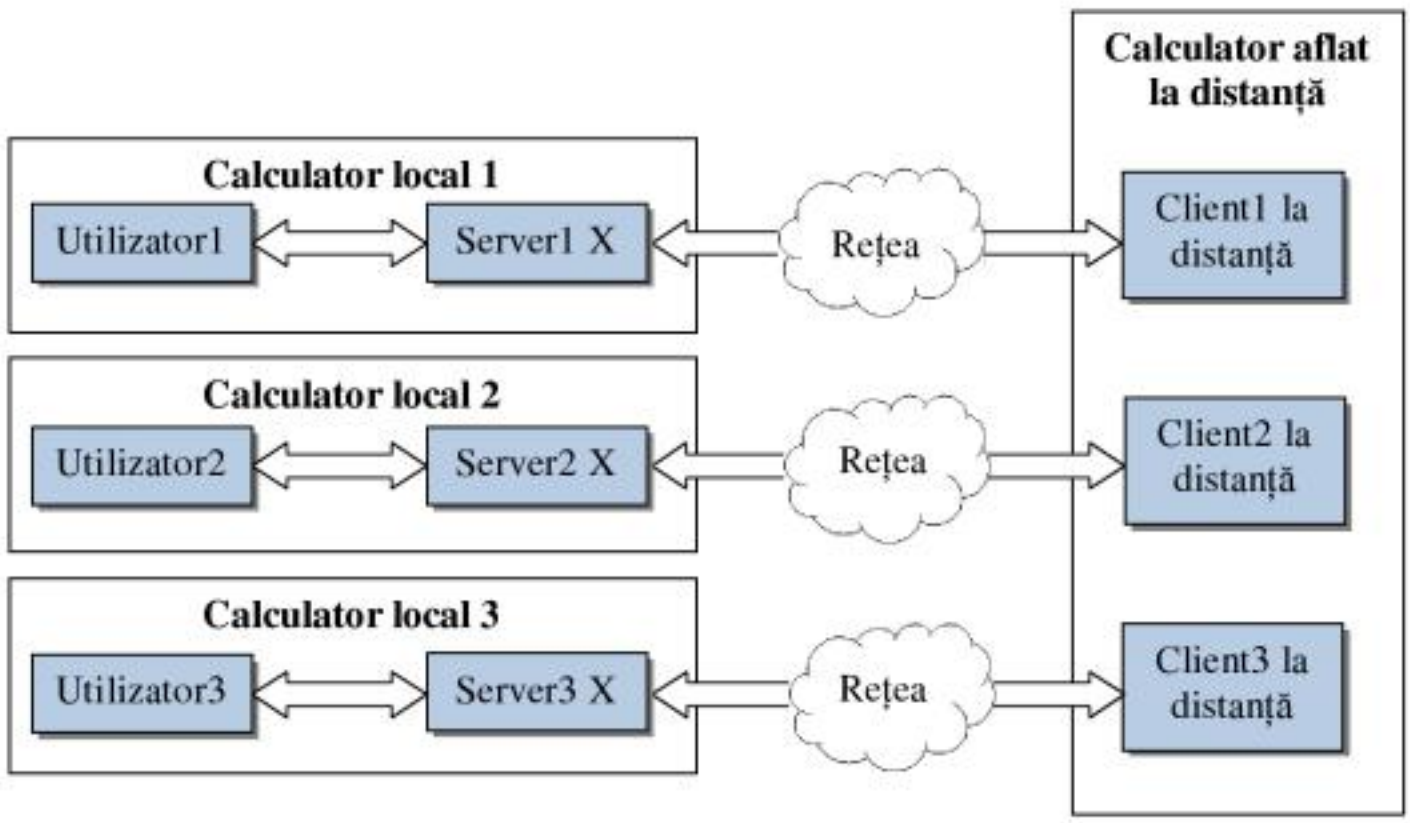


Figura 12-4 Conectarea client - server în cadrul protocolului X

În exemplul de mai sus, pe un calculator aflat la distanță rulează trei clienți X. Acest calculator poate sa nu aibă conectat nici un monitor şi nici mouse sau tastatură. Fiecare dintre cei trei clienți este conectat la un server X aflat pe un calculator local. Serverul X de pe calculatorul local preia date de la utilizator (de exemplu prin intermediul tastaturii) şi le transferă clienților, primind înapoi de la aceştia informații care trebuie prezentate utilizatorului pe ecran. Toată procesarea informației de la utilizator este realizată de calculatorul aflat la distanță.

X nu conține nici o specificație legată de felul în care arată interfața cu utilizatorul (*Application User Interface*): de exemplu modul în care arată butoanele, meniurile, ferestrele, etc. Aceste specificații sunt realizate în schimb de către alte componente, cum ar fi *window manager*, *desktop environment* sau *application-specific GUI* care se găsesc între X Server şi ecran.

Un **window manager** controlează amplasarea şi modul în care arată ferestrele aplicațiilor. Acesta este un tip special de client X (se conectează la serverul X ca orice alt client) care controlează felul în care arată ceilalți clienți X. *Window managerul* face ca toți clienții să aibă caracteristici comune: bara de titlu a ferestrei, butoanele de *minimize* şi *maximize*, etc

El poate crea o interfață asemănătoare cu cea din Microsoft Windows sau cu cea din Mac OS X. *Window managerele* pot oferi funcționalități de bază (cum ar fi twm, *window managerul* care vine împreună cu X) sau funcționalități complexe (cum ar fi Enlightenment ce se apropie ca funcționalitate de un *desktop environment*). Exemple: Kwin este *window managerul* folosit de KDE, Metacity cel folosit de GNOME şi Compiz este cel utilizat pentru desenarea interfeței 3D din proiectul cu acelaşi nume.

În cazul în care această componentă lipseşte, se vor putea crea ferestre noi dar nu vor avea bara de titlu. În plus, ele nu vor putea fi mutate pe ecran, nu vor putea fi închise cu ALT+F4, nu va funcționa ALT+TAB, nu se va putea muta o fereastră din fundal (*background*) în prim-plan (*foreground*). În schimb conținutul ferestrelor va rămâne neschimbat.

Stilul în care arată ferestrele (aspectul lor vizual) este controlat de o componentă a *window managerului*, numită **window decorator**. Un exemplu cunoscut este emerald, *window decorator-ul* folosit de Beryl sau Compiz.

Un **desktop environment** include un *window manager*, mai multe aplicații și o interfață consistentă din punct de vedere vizual. GNOME și KDE sunt două exemple de *desktop environment*.

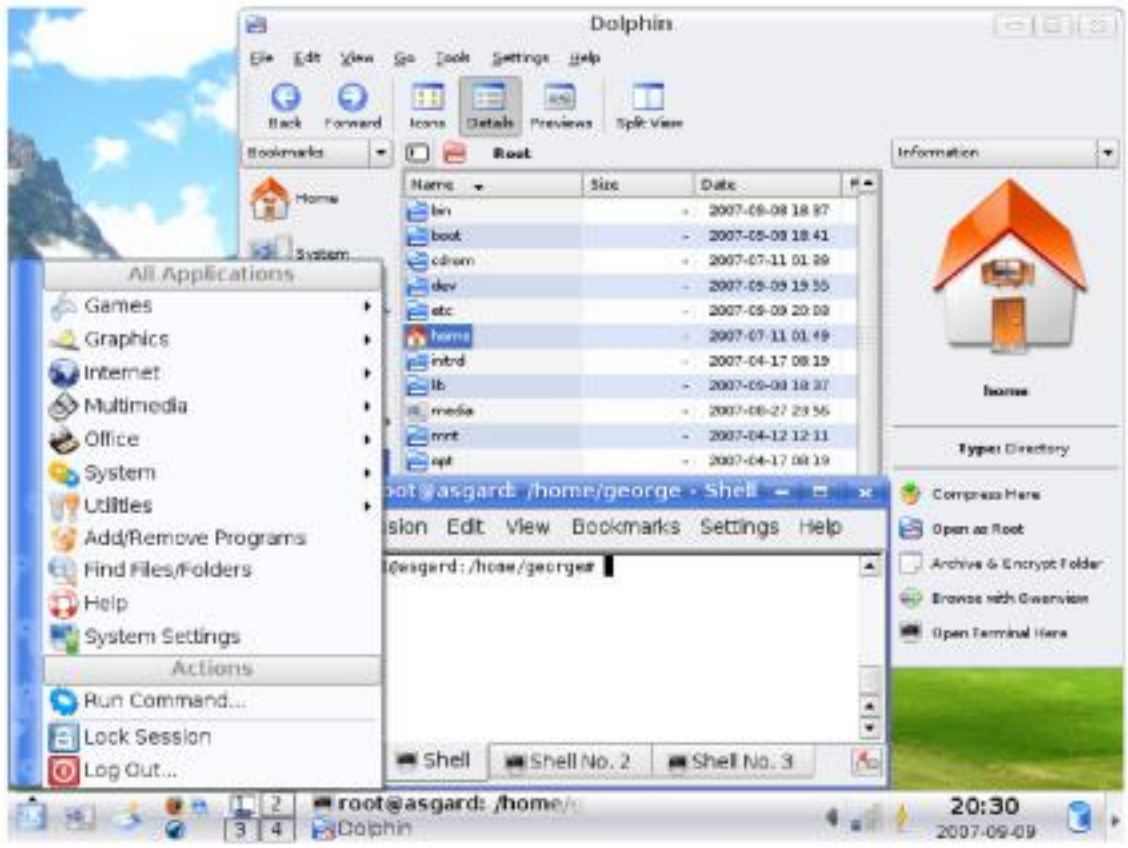


Figura 12-5 KDE

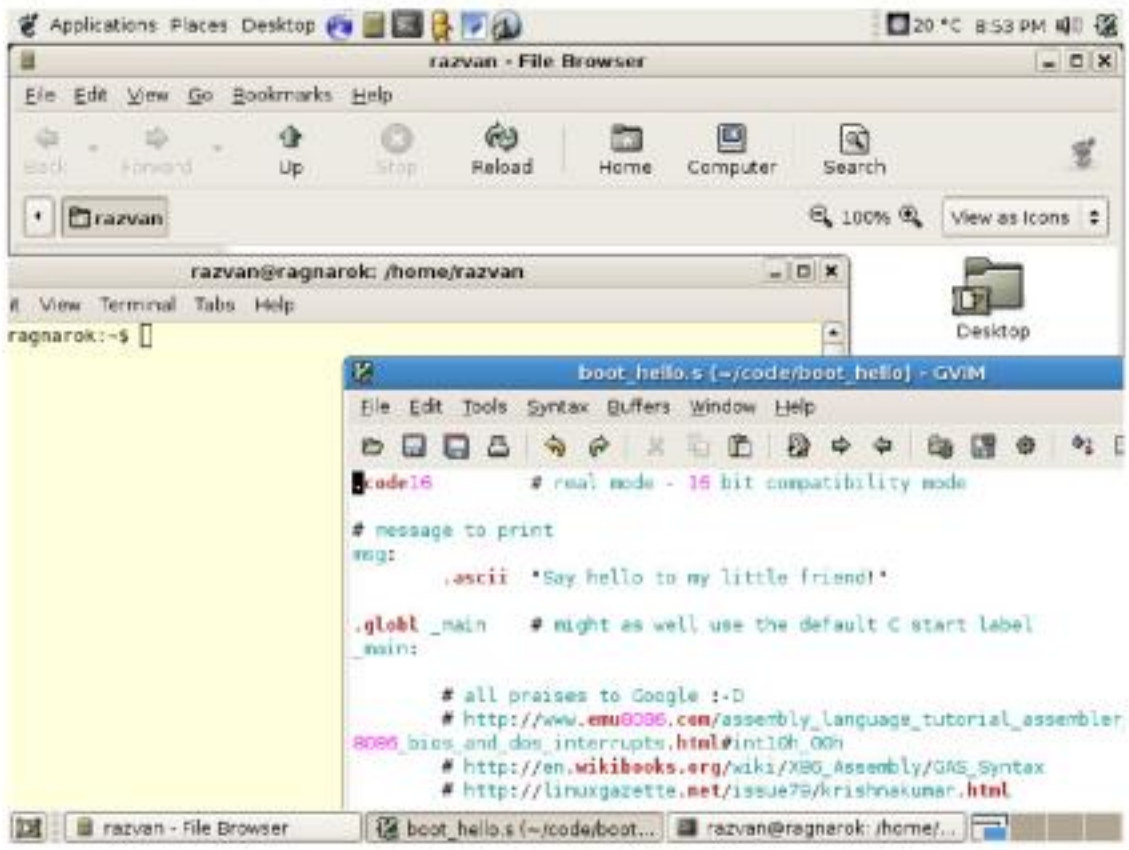


Figura 12-6 GNOME

Un **display manager** este un program care permite pornirea unei sesiuni pe un server X de pe același calculator sau de pe unul aflat la distanță. Acesta este implementat ca un tip special de client X. *Display managerul* afișează utilizatorului un ecran de autentificare (*login*) permițându-i acestuia să introducă un nume de utilizator și o parolă. După ce autentificarea s-a realizat cu succes o nouă sesiune este pornită. Exemplu de *display managere*: kdm folosit de KDE, gdm folosit de GNOME sau xdm, *display managerul* implicit din X Window System.

Figura 12-7 Fereastra de login implicită din kubuntu (kdm)

Figura 12-8 Fereastra de login implicită din ubuntu. (gdm)

Atunci când *display managerul* rulează pe acelaşi calculator cu serverul X, el va porni serverul înainte de afişa fereastra de autentificare. Astfel *display-managerul* are o funcție asemănătoare cu procesele login şi getty. [referință la getty]

Atunci când *display managerul* rulează pe un calculator aflat la distanță, el se comportă ca un server de telnet, cerând utilizatorului un nume de utilizator şi o parolă şi pornind apoi o sesiune cu serverul X.

12.1.3 Evoluția serverului X

X.Org şi XFree96 sunt cele mai des întâlnite implementări ale serverului X.

XFree86 a apărut în 1992. El provine din serverul X386 pentru IBM PC (care se baza pe o versiune a protocolului X). XFree86 a evoluat în timp de la un simplu server X la cea mai populară implementare a X-ului în anii '90 şi la începutul anilor 2000. El a reprezentat un standard *de facto* în care erau implementate toate inovațiile.

În mai 1999 Open Group a format **X.Org**, rezultând apariția X11R6.5.1. Până la începutul lui 2003, în timp ce popularitatea Linux a crescut, X.Org a rămas inactiv, majoritatea inovațiilor și dezvoltărilor fiind realizate de XFree86.

În martie 2004, după anumite rupturi în cadrul XFree86, este lansată o nouă versiune sub o licență mult mai strictă decât cea utilizată până în acel moment, pe care *Free Software Foudation* a considerat-o incompatibilă cu licența GPL.

Noua licență a devenit inacceptabilă pentru multe din proiectele care depindeau de X. Din această cauză au apărut multe ramuri de server X care se trăgeau din XFree86, ramuri care rămâneau în continuare sub licența GPL.

La începutul lui 2004 X.Org și freedesktop.org au format X.Org Foundation. Imediat după, în conjuncție cu problemele de licență pe care le avea XFree86, versiunea X116.5 a X.Org a fost combinată cu versiunea 4.4RC2 a XFree86 și a rezultat versiunea X116.6 a X.Org. Aceasta a fost prima versiune de X.Org apărută sub licența X.Org Foundation.

Din acel moment X.Org a devenit cea mai utilizată implementare a serverului X.

12.2 Pornirea și oprirea interfeței grafice

În cazul în care interfața grafică nu pornește o dată cu sistemul de operare, ea poate fi pornită separat. Pentru a realiza acest lucru există mai multe posibilități:

Folosind xinit

`xinit` este un program care pornește serverul X și un prim client care se conectează la el. Utilizatorul nu mai trebuie sa se autentifice într-o fereastră de login deoarece el s-a autentificat în sistem înainte de a rula comandă `xinit`. Atunci când acest prim client își încheie activitatea, `xinit` va opri serverul X:

```
george@asgard:~$ man xinit
[...]
The xinit program is used to start the X Window System server and a first
client program on systems that cannot start X directly from /etc/init or in
environments that use multiple window systems.  When this first client exits,
xinit will kill the X server and then terminate.
```

Folosind startx

`startx` este un front-end pentru `xinit` care oferă utilizatorului o interfață mai complexă:

```
george@asgard:~$ man startx
[...]
The startx script is a front end to xinit that provides a somewhat nicer
user interface for running a single session of the X Window System. It is
often run with no arguments.
```

Pornind un Display Manager

Atunci când un Display Manager este pornit pe calculatorul local, el va porni mai întâi un server X, urmând ca apoi să prezinte utilizatorului un ecran de autentificare.

Pornirea unui Display Manager se poate face în modul următor:

- pentru kdm:

```
asgard:/home/george# /etc/init.d/kdm start
```

- pentru gdm:

```
asgard:/home/george# /etc/init.d/gdm start
```

Pentru oprirea interfeței grafice este suficientă oprirea *display managerului*. Acest lucru se poate realiza în felul următor:

```
asgard:/home/george# kill -9 *dm
```

Comanda anterioară oprește toate procesele al căror nume este oricât de lung și se termină cu șirul de caractere „dm".

O alta metodă de oprire a interfeței grafice este rularea scripturilor `kdm` sau `gdm` cu paramerul `stop`:

```
asgard:/home/george# /etc/init.d/kdm stop
```

respectiv

```
asgard:/home/george# /etc/init.d/gdm stop
```

Pentru oprirea și repornirea imediată a interfeței grafice este suficient să fie repornit *display managerul* folosind tot scripturile `kdm` și `gdm` însă cu parametrul `restart`:

```
asgard:/home/george# /etc/init.d/kdm restart
```

respectiv

```
asgard:/home/george# /etc/init.d/gdm restart
```

De asemenea oprirea și repornirea imediată a serverului X se poate realiza și prin combinația de taste Ctrl+Alt+Backspace.

12.3 Configurarea serverului X

12.3.1 Secțiunile din xorg.conf

Configurarea serverului X se realizează printr-un fișier text. Pentru X.Org acest fișier este `/etc/X11/xorg.conf`. În cazul lui XFree86 fișierul se numește `/etc/X11/XF86Config`. Cele două fișiere au o structura asemănătoare, diferențele dintre ele fiind destul de mici.

Acest fișier este organizat în diferite secțiuni, fiecare dintre ele putându-se împărți la rândul ei în subsecțiuni.

Un exemplu de fișier `xorg.conf` este următorul:

```
Section "Files"
        FontPath        "/usr/share/fonts/X11/100dpi"
        FontPath        "/usr/share/fonts/X11/75dpi"
        [...]
EndSection

Section "Module"
        Load    "bitmap"
        Load    "dri"
        Load    "glx"
        Load    "int10"
        [...]
EndSection

Section "InputDevice"
        Identifier      "Generic Keyboard"
        Driver          "kbd"
        Option          "CoreKeyboard"
        Option          "XkbRules"      "xorg"
        Option          "XkbModel"      "pc105"
        Option          "XkbLayout"     "us"
EndSection

Section "InputDevice"
```

```
        Identifier     "Configured Mouse"
        Driver         "mouse"
        Option         "CorePointer"
        Option         "Device"               "/dev/input/mice"
        Option         "Protocol"             "ImPS/2"
        Option         "ZAxisMapping"         "4 5"
        Option         "Emulate3Buttons"      "true"
EndSection

[...]

Section "Device"
        Identifier     "Intel Corporation Mobile 945GM/GMS"
        Driver         "intel"
        BusID          "PCI:0:2:0"
EndSection

Section "Monitor"
        Identifier     "Generic Monitor"
        Option         "DPMS"
        HorizSync      28-64
        VertRefresh    43-60
EndSection

Section "Screen"
        Identifier     "Default Screen"
        Device         "Intel Corporation Mobile 945GM/GMS"
        Monitor        "Generic Monitor"
        DefaultDepth   24
        SubSection "Display"
                Depth          4
                Modes          "1280x800" "1280x768" "1024x768" "800x600"
        EndSubSection
        SubSection "Display"
                Depth          16
                Modes          "1280x800" "1280x768" "1024x768" "800x600"
        EndSubSection
        SubSection "Display"
                Depth          24
                Modes          "1280x800" "1280x768" "1024x768" "800x600"
        EndSubSection
EndSection

Section "ServerLayout"
        Identifier     "Default Layout"
        Screen         "Default Screen"
        InputDevice    "Generic Keyboard"
        InputDevice    "Configured Mouse"
        InputDevice    "Synaptics Touchpad"
        [...]
EndSection
[...]
```

Principalele secțiuni ale fișierului de configurare sunt:

- `Files`
- `Module`
- `InputDevice (Keyboard și Mouse)`
- `Device`
- `Monitor`
- `Screen`
- `ServerLayout`

Comun unora dintre aceste secțiuni este câmpul *Identifier*. Acesta specifică numele dispozitivului care urmează să fie configurat în secțiunea respectivă. Acest nume este ales de utilizator, fiind recomandat ca el ca fie cat se poate de clar în privința dispozitivului asociat.

Secțiunea **Files** e folosită pentru a specifica serverului X calea către anumite fișiere de care are nevoie (cel mai uzual în această secțiune se specifică fonturile pe care le folosește serverul X). Unele dintre aceste căi pot fi setate de asemenea din linie de comandă. Secțiunea Files este opțională, la fel cum sunt și intrările care apar în ea.

Secțiunea **Module** e folosită pentru a specifica ce module ale serverului X să vor fi încărcate. Acest lucru se face folosind sintaxa: `Load modulename`.

Secțiunea **InputDevice** specifică configurările pentru dispozitivele de intrare (*input*). În această categorie intră, printre altele, tastatura, mouse-ul, *stylus-ul*, *touch screen-ul*. Două dintre opțiunile utile prezente în cazul tastaturii sunt specificarea tipului de tastatură (de exemplu tastatură cu 104 sau cu 105 taste) şi specificarea modului în care sunt aşezate tastele: ca la o tastatură în limba română sau în limba engleză, etc.

În cazul mouse-ului, opțiunea ZAxisMapping este cea răspunzătoare de funcționarea scroll-ului (se asociază funcțiile de *scroll up* şi *scroll down* pe butoanele 4 şi 5 ale mouse-ului).

Secțiunea **Device** conține configurările plăcii video. Cele mai importante configurări din această secțiune sunt cele care specifică driverul folosit şi locația plăcii video (`BusID`).

O alta secțiune importantă este secțiunea **Monitor**. Aici se pot specifica frecvențele orizontală şi verticală de funcționare a monitorului. În cazul monitoarelor CRT, dacă monitorul suportă o frecvența verticală mai mare de 60Hz şi aceasta nu a fost configurată automat, trebuie introduse în aceasta secțiune specificațiile monitorului (frecvența orizontală şi verticală).

Secțiunea **Screen** asociază un monitor cu o placa video, ambele definite anterior. Aceasta secțiune conține subsecțiuni în care sunt definite perechi de adâncimi de culoare şi de rezoluții pentru care adâncimile sunt disponibile. Alegerea unei adâncimi de culoare şi a unei rezoluții este făcuta de către serverul X în ordinea descrescătoare a adâncimii de culoare, şi pentru fiecare adâncime de culoare în ordinea în care sunt definite rezoluțiile. Adâncimea de culoare specificată prin `DefaultDepth` este cea încercată în mod implicit. Dacă o anumită configurație nu poate fi aplicată, se trece la următoarea valoare definită în fişier.

12.4 Configurarea sistemului din KDE

În Kubuntu toate configurările importante ale sistemului sunt reunite într-un panou de comandă numit *System Settings*. Acest panou este asemănător ca funcționalitate cu Control Panel din Windows şi poate fi accesat din meniul de Start.

12.4.1 System Settings

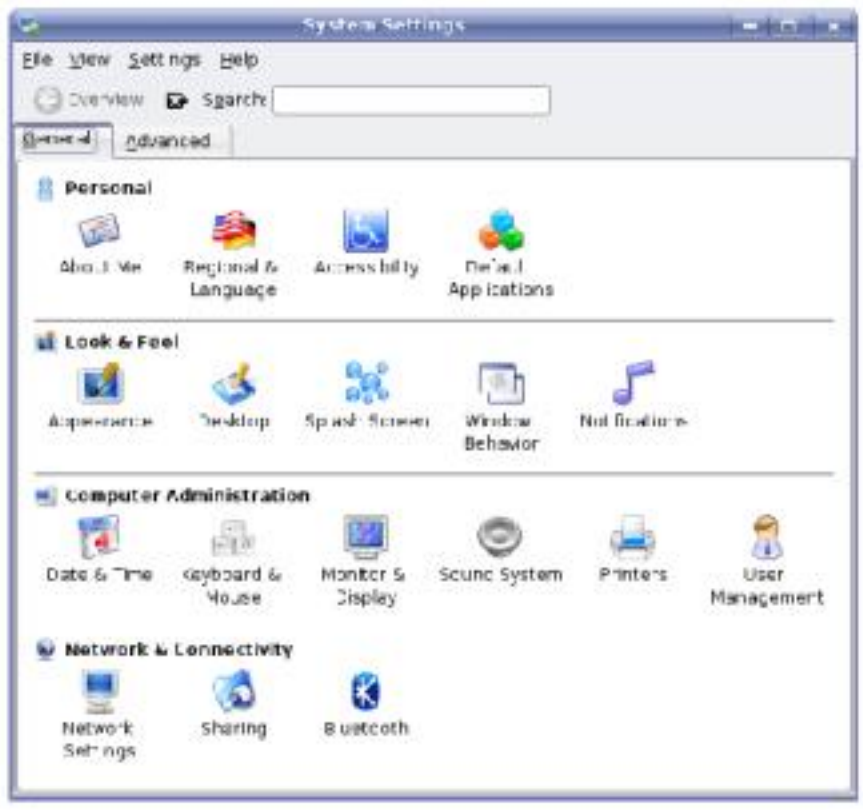


Figura 12-9 Sytem settings pentru Kubuntu Feisty

Figura Figura 12-9 este prezentată fereastra de configurare pentru Kubuntu Feisty (7.04). Aceasta oferă acces la principalele configurări ale sistemului de operare: managementul utilizatorilor, configurări de aspect vizual, configurarea perifericelor, configurarea rețelei, etc.

12.4.2 Configurări de bază ale sistemului de operare

About Me: permite introducerea datelor personale ale utilizatorului şi specificarea căii către directoarele sale personale: directorul *home*, directorul unde se află fişierele de pe Desktop şi directorul Autostart. În directorul de Autostart se pot pune scripturi care vor fi rulate la pornirea KDE.

Regional & Language conține configurări legate de regiunea în care se găseşte utilizatorul (de exemplu modul în care sunt afişate numerele, separatorul pentru partea zecimală a numerelor, simbolul pentru moneda utilizată, tipul de calendar folosit, formatul orei şi al datei). Tot în această secțiune se poate alege tipul de tastatură (de exemplu cu 104 sau 105 taste) şi felul în care sunt aşezate tastele (de exemplu ca pentru o tastatură în limba română sau ca pentru una în limba engleză).

Accesibility oferă acces la diferire configurări pentru persoanele cu dizabilități.

Default Applications permite specificarea aplicațiilor folosite la deschiderea diferitelor tipuri de fişiere.

12.4.3 Schimbarea aspectului interfeței grafice

Appearance conține configurări legate de modul în care arată interfața grafică. Se poate alege schema de culori a ferestrelor şi butoanelor (*Colors*), fonturile folosite pentru afişarea textelor din interfața grafică (*Fonts*), simbolurile folosite în sistem (*Icons*), stilul în care arată elementele vizuale (de exemplu butoanele pot fi făcute mai rotunde sau mai dreptunghiulare) (*Style*), felul în care arată bara de titlu a ferestrelor (*Window Decorations*)

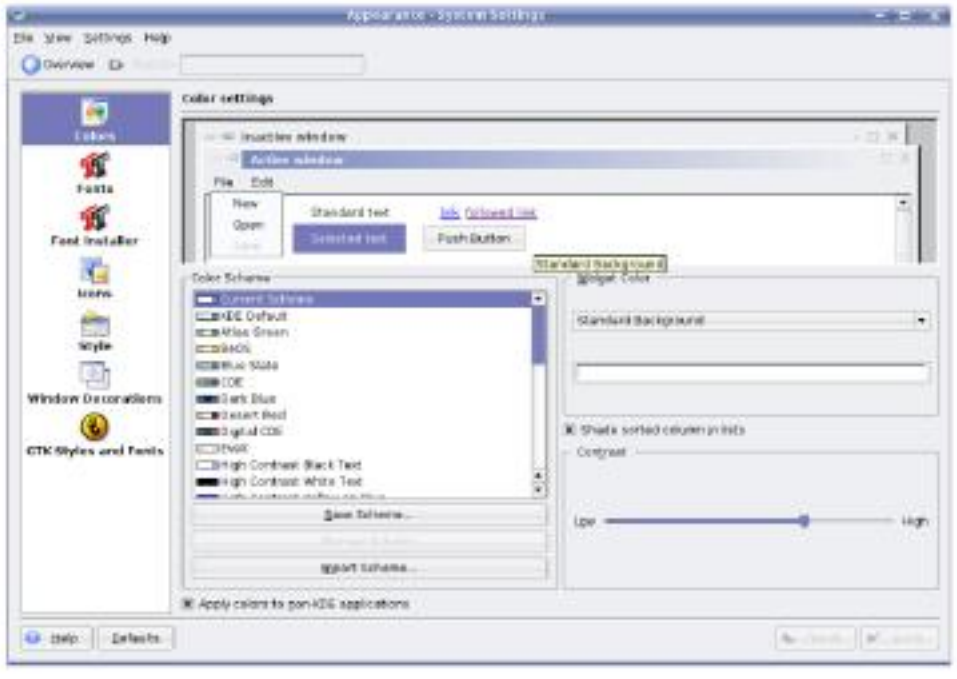


Figura 12-10 Alegerea schemei de culori

Desktop permite modificarea fundalului (*Background*), a screen saver-ului (*Screen Saver*) sau a numărului de desktop-uri (Multiple Desktops). În mod normal în Linux se folosesc două sau patru desktop-uri. Fiecare poate avea propria imagine de fundal, propriile simboluri asociate unor programe, şi propriile programe deschise.

O practică obişnuită legată de utilizarea mai multor desktop-uri este aceea de a grupa programele pe desktop-uri funcție de utilitate. De exemplu, pe desktop-ul 1 se poate deschide

un web browser, pe desktop-ul doi un client de email, pe desktop-ul 3 o temă de casă şi pe desktop-ul 4 un program de mesagerie instant. Avantajul oferit de această abordare este acela că fiecare desktop devine mai liber, având mai puține programe deschise.

Splash Screen permite alegerea animației ce va fi afişată în timp de porneşte KDE şi care arată stadiul curent al procesului de înărcare.

Dintre configurările prezente în **Window Behaviour**, cea mai importantă este cea care defineşte acțiunea petrecută atunci când se realizează un dublu click pe bara de titlu a unei ferestre. Implicit fereastra va realiza un *roll-up*, adică se va minimiza şi practic se va restrânge în spațiul ocupat de bata de titlu. O alta acțiune care se poate configura este aceea ca, după dublu click, fereastra să se maximizeze (comportament specific Windows-ului).

Notifications conține configurări legate de sunetele asociate diferitelor acțiuni din sistemul de operare (*System Notifications*). De asemenea tot în această secțiune se pot defini şi aplicațiile care pornesc atunci când un disc este inserat în unitatea optică (echivalentul opțiunii autorun din Windows) (*Storage Media Notifications*).

12.4.4 Configurări administrative

Date & Time permite configurarea datei şi orei sistemului. Tot aici se poate alege şi fusul orar corespunzător zonei în care se găseşte utilizatorul.

Keyboard & Mouse conține configurări legate de tastatură şi mouse. Una din cele mai importante opțiuni prezente aici este cea legată de starea NumLock la inițializarea KDE. Se poate alege ca NumLock să fie activat, dezactivat sau să îşi păstreze starea anterioară. În cadrul subsecțiunii *Mouse* se poate alege tema cursorului de la mouse.

Tot în această secțiune se pot configura combinațiile de taste (*shortcut*) pentru diferite funcții din KDE. Cele mai utilizate combinații (cu valorile lor implicite) sunt:

Combinația de taste	Acțiune
Alt + F1	Deschide fereastra de ajutor
Alt + F2	Deschide fereastra de Run de unde se poate porni o aplicație specificând numele executabilului
Alt + Esc sau Ctrl + Esc	Porneşte aplicația KSysGuard în care este afişată o listă a proceselor active
Ctrl + Fn	Se face trecerea la desktop-ul n. n este un număr între 1 şi 8
Alt + Tab sau Alt + Shift + Tab	Se trece de la o aplicație la alta. Folosind tasta Shift se ciclează aplicațiile în ordine inversă
Ctrl + Tab sau Ctrl + Shift + Tab	Realizează ciclarea între desktop-uri. În cazul în care se foloseşte tasta Shift se ciclează desktop-urile în ordine inversă
Alt + F4	Închide fereastra activă
Ctrl + Alt +Esc	Această combinație permite terminarea unei aplicații în mod forțat. După ce combinația de taste a fost efectuată, cursorul mouse-ului se va transforma (cel mai uzual într-un X) şi fereastra pe care se va realiza un click va fi terminată în mod forțat. Funcționarea aplicației este similară comenzii kill
Ctrl + Alt + Backspace	Reporneşte serverul X şi implicit termină sesiunea curentă de KDE
Ctrl + Alt + „Numpad+” (tasta „+” de pe Numpad)	Schimbă rezoluția ecranului la următoarea rezoluție disponibilă (în cazul în care în fişierul xorg.conf sau XFree86.conf au fost specificate mai multe rezoluții disponibile)
Ctrl + Alt + „Numpad-” (tasta „-” de pe Numpad)	Schimbă rezoluția ecranului la rezoluția anterioară disponibilă (în cazul în care în fişierul xorg.conf sau XFree86.conf au fost specificate mai multe rezoluții disponibile)

Tabelul 12.1 Combinațiile de taste uzuale în KDE

Monitor & Display permite, printre altele, alegerea uneia dintre rezoluțiile de ecran disponibile şi a parametrilor de culoare (gamma, luminozitate, contrast).

Sound System conține configurări ale plăcii audio. În tabul *Advanced* se poate alege driverul folosit pentru placa audio. Cel mai folosit driver pentru placa audio este cel care face parte din proiectul ALSA (*Advanced Linux Sound Architecture*).

12.4.5 Configurarea rețelei

Network settings permite introducerea parametrilor de rețea.

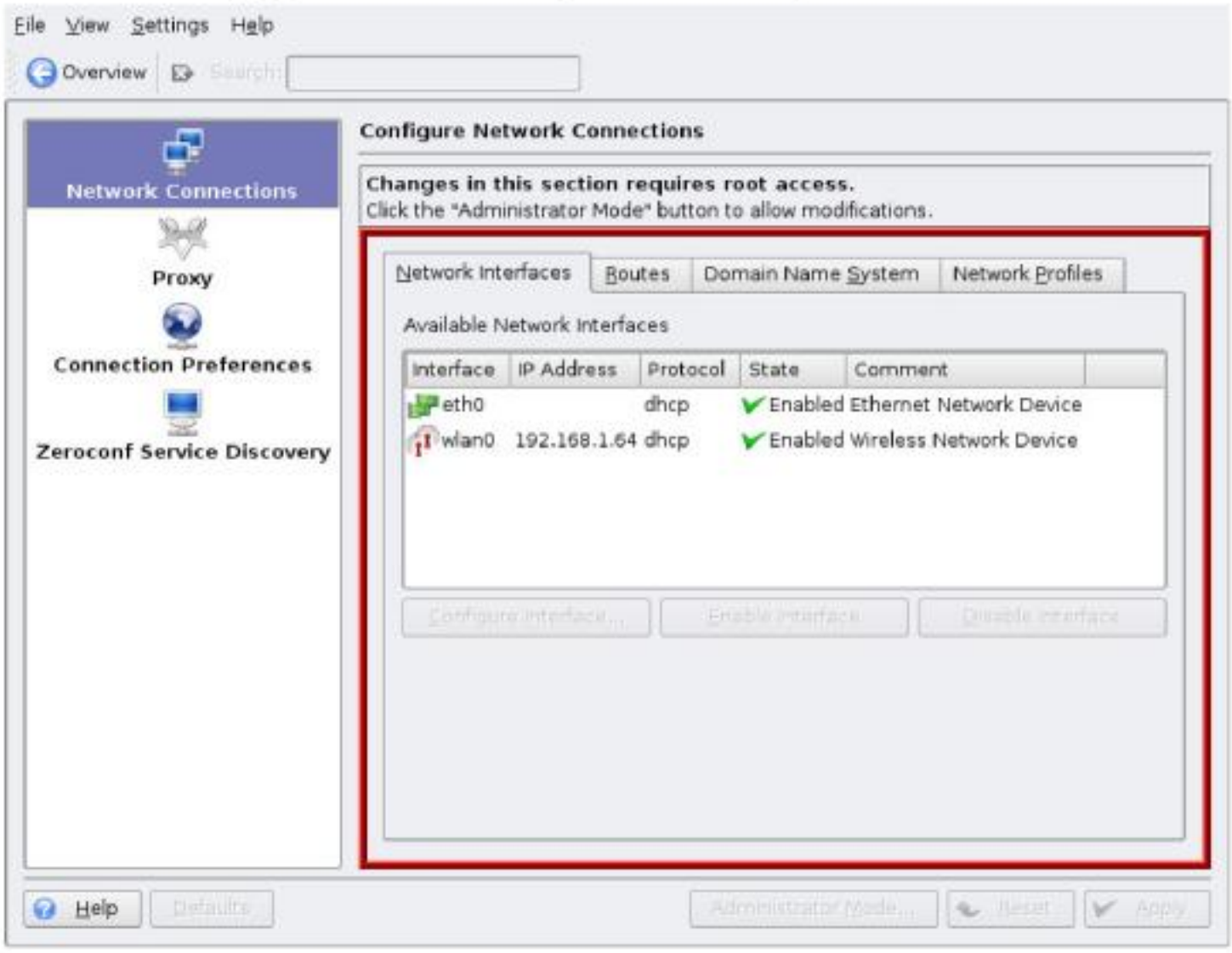


Figura 12-11 Configurarea parametrilor de rețea

În tabul **Network Interfaces** apare lista interfețelor disponibile. Alegând una din aceste interfețe se pot configura parametrii ei [vezi 7.2] (pentru a putea configura şi adresa de broadcast şi pe cea a gateway-ului trebuie apăsat butonul *Advanced*):

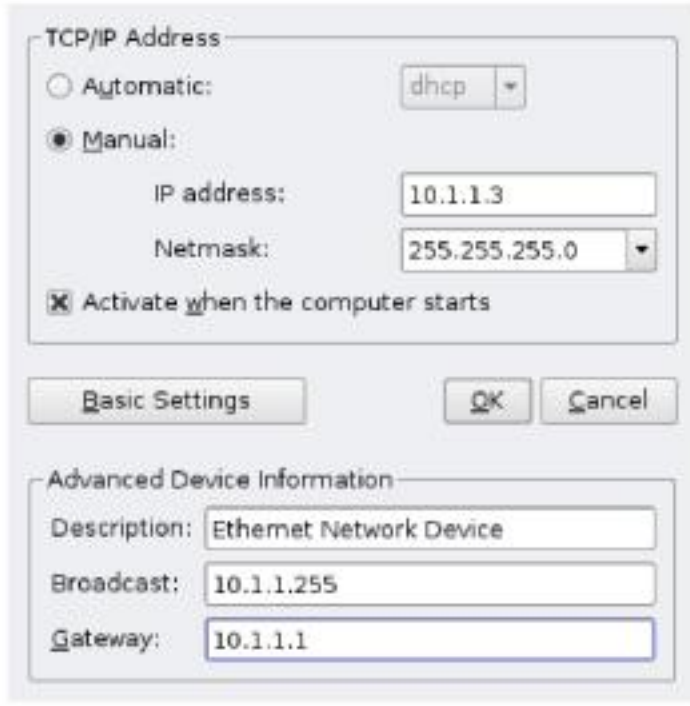


Figura 12-12 Configurarea parametrilor de rețea de bază

Tabul **Routes** permite configurarea rutelor şi a gateway-ului implicit.

Tabul **Domain Name System** conține atât configurări ale serverelor de DNS cât şi administrarea aliasurilor. De asemenea tot aici se poate configura numele stației (*hostname*):

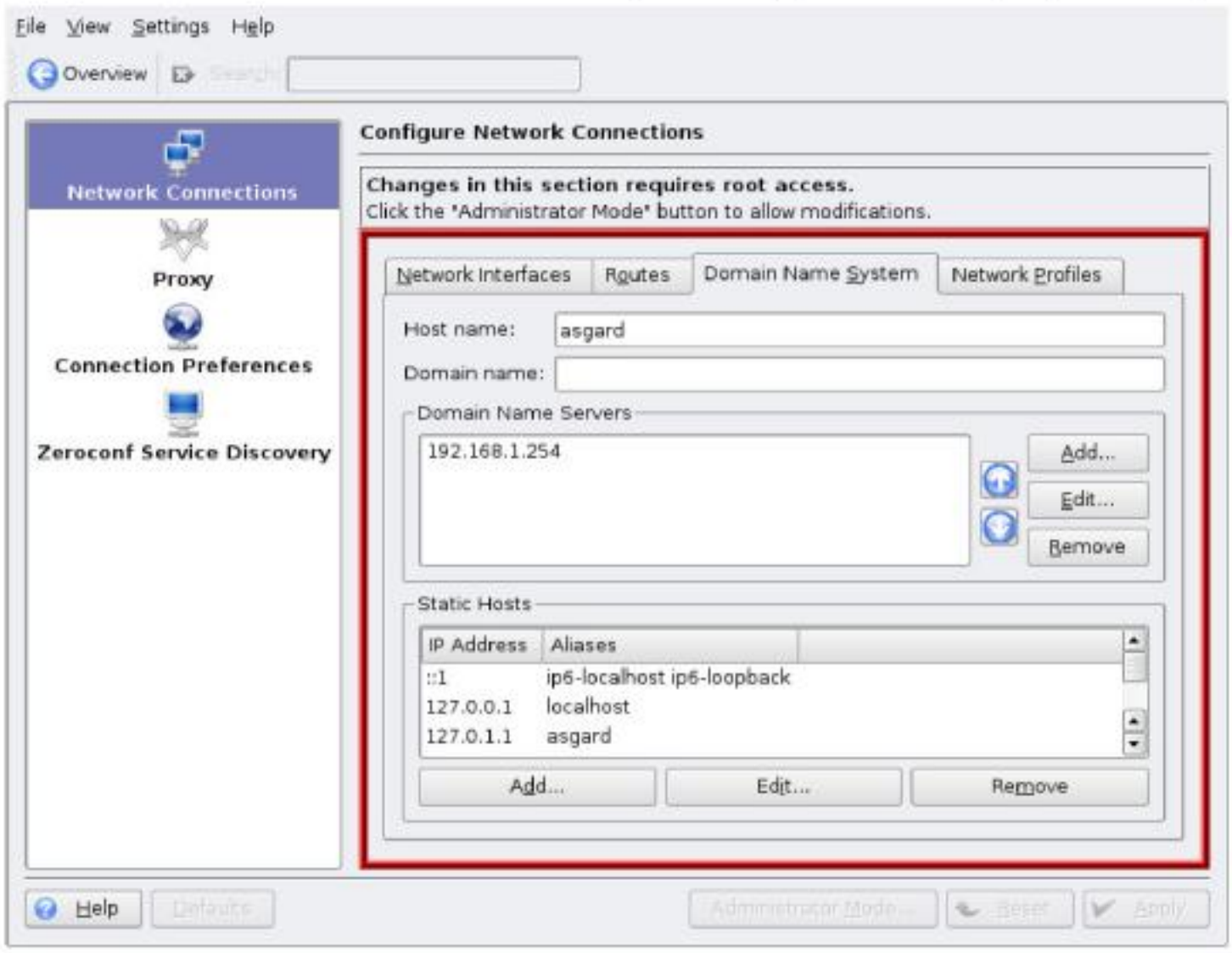


Figura 12-13 Configurarea serverelor de DNS

Pentru a configura interfața de rețea folosind DHCP, se apele opțiunea *Automatic*. Pentru a realiza configurările static, se alege opțiunea *Manual*. Astfel se pot configura adresa IP, masca

de rețea, adresa de broadcast şi adresa gateway-ului. De asemenea, se poate adăuga interfeței de rețea şi o descriere.

12.4.6 Managementul utilizatorilor

Configurarea utilizatorilor se realizează în secțiunea **User Management**:

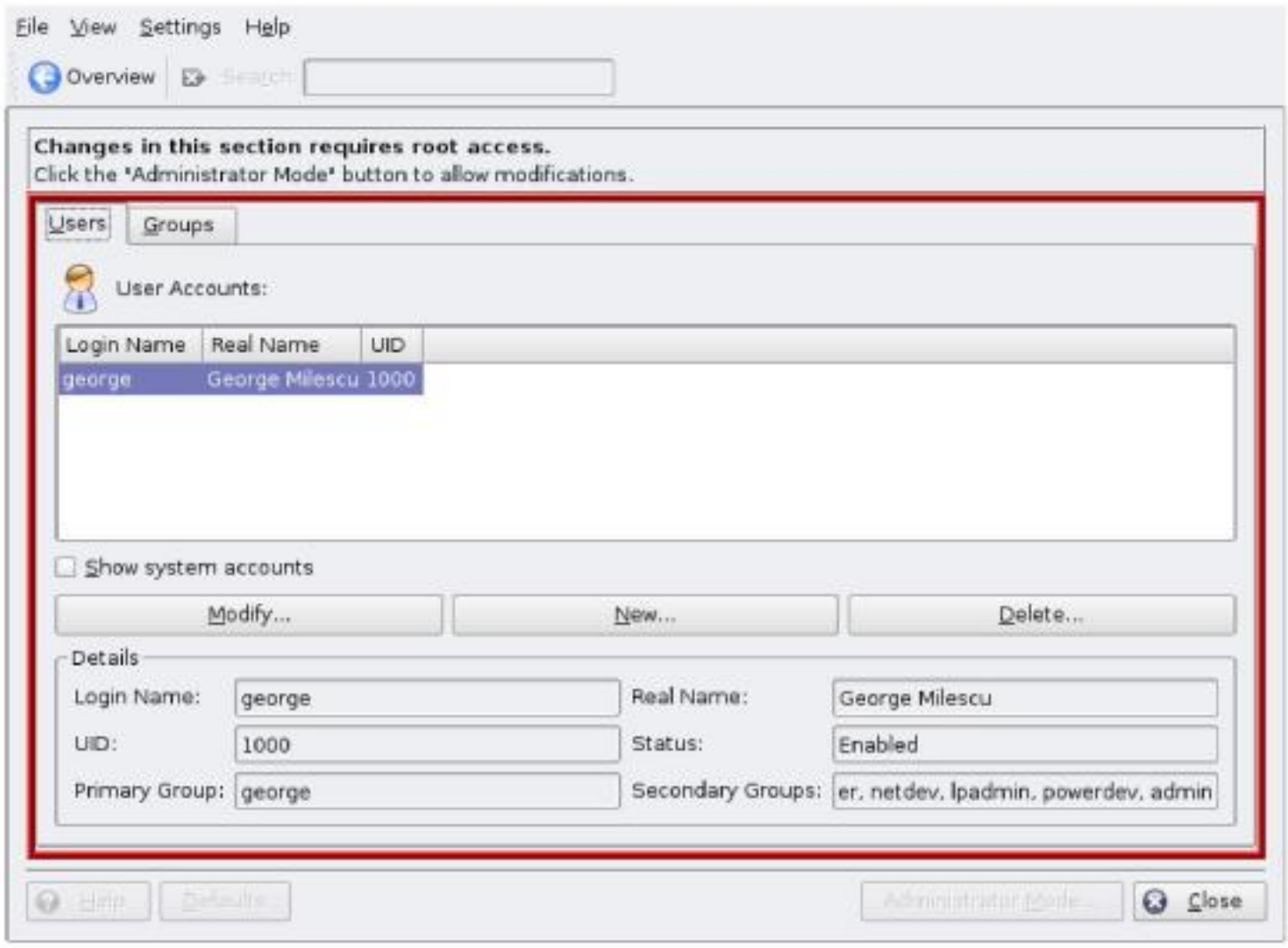


Figura 12-14 Administararea utilizatorilor

Aici se pot adăuga, şterge sau modifica utilizatorii din sistem. Se pot modifica numele de login, numele real al utilizatorului, grupurile din care face parte, etc. Pentru a modifica datele unui utilizator se selectează utilizatorul dorit şi se apasă butonul *Modify*. Fereastra afişată va fi următoarea:

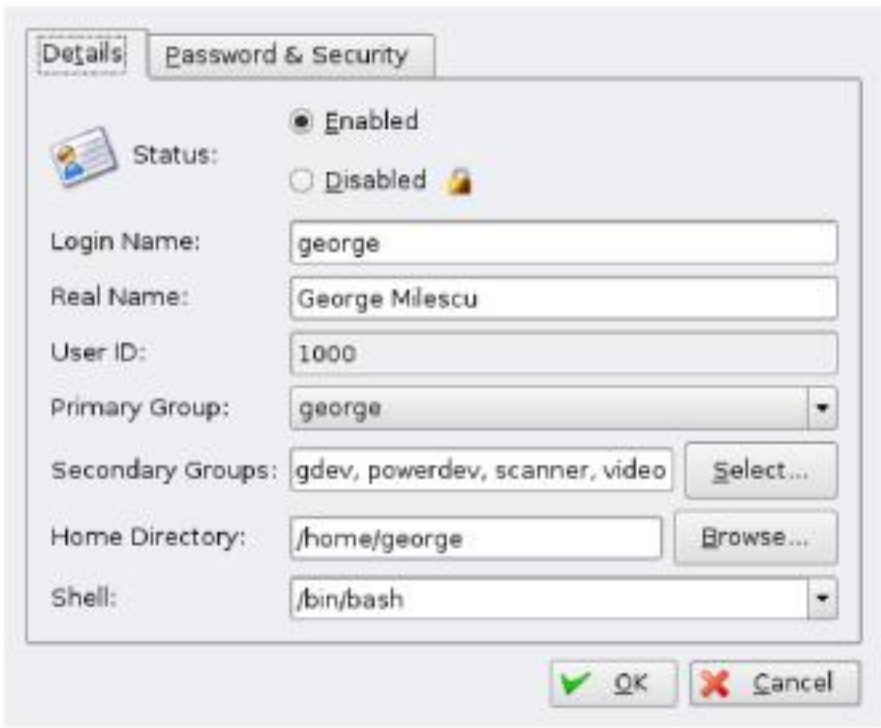


Figura 12-15 Modificarea datelor unui utilizaor

12.4.7 Monitorizarea sistemului şi managementul proceselor

Pentru monitorizarea unui sistem din interfaţa grafică pot fi folosite mai multe aplicaţii. Dintre acestea, cea mai uzuală (în KDE) este KSysGuard. Aceasta vine implicit împreună cu Kubuntu. Pentru pornirea ei se alege meniul Start | System | KSysGuard.

Această aplicaţie are două componente principale: cea de monitorizare a sistemului şi cea de management al proceselor.

Pentru monitorizarea sistemului KSysGuard pune la dispoziţie o colecţie de senzori ale căror valori pot fi urmărite sub formă de grafice:

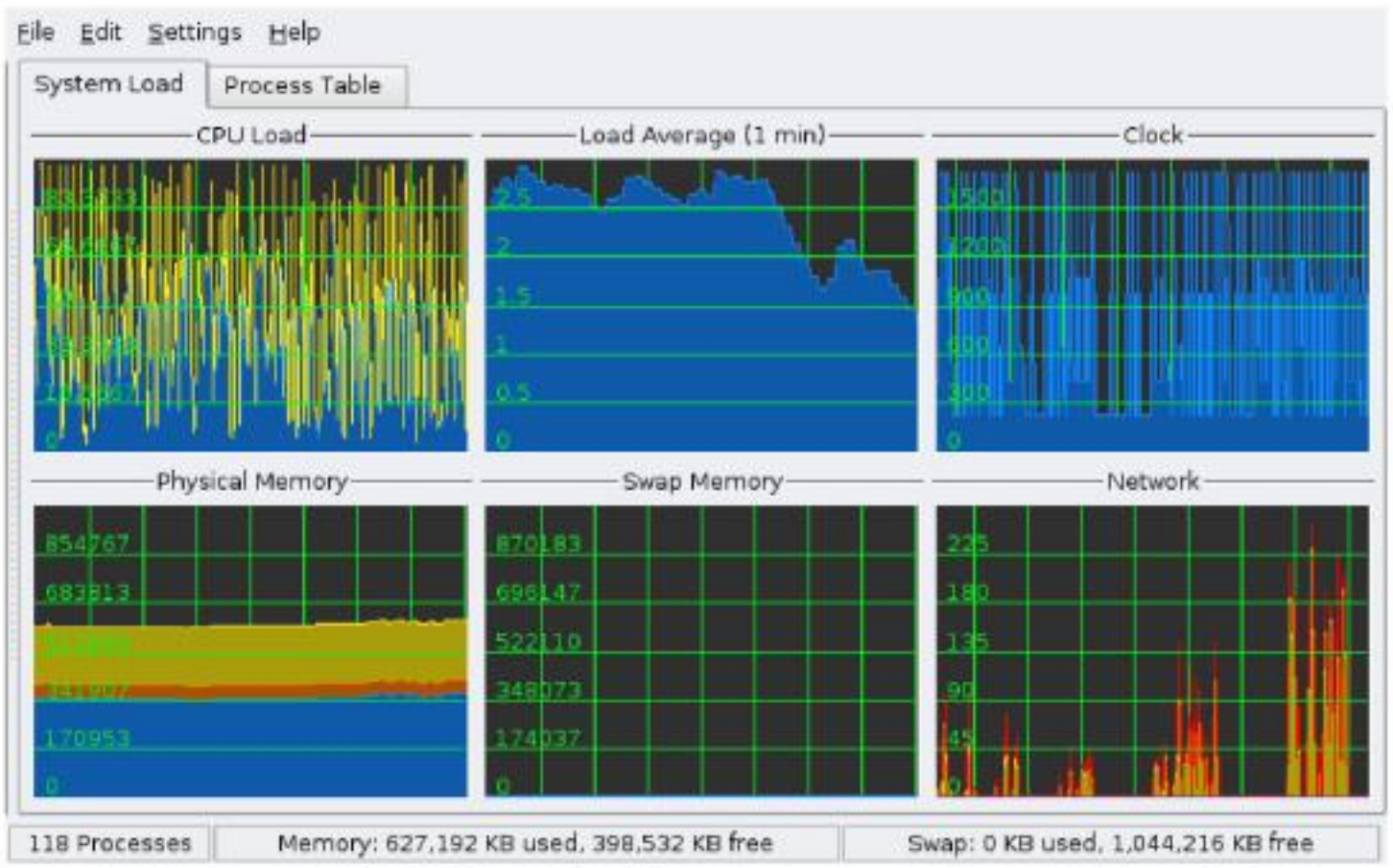


Figura 12-16 Monitorizarea sistemului

Paleta de senzori disponibili este foarte mare. Pe un grafic pot fi plasați mai mulți senzori pentru a urmări evoluția lor în paralel. De exemplu, pentru o interfață de rețea, senzorii disponibili sunt următorii:

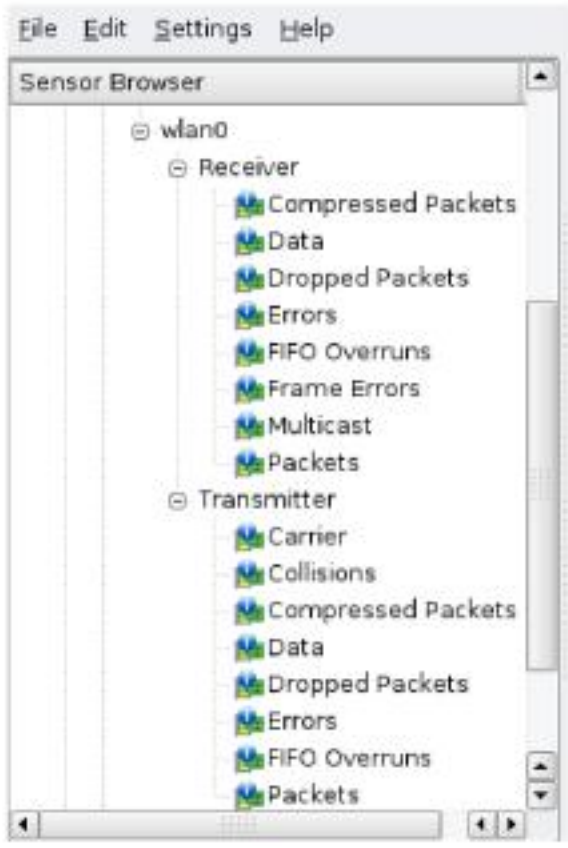


Figura 12-17 Senzorii disponibili pentru o interfață de rețea

Managementul proceselor se realizează prin intermediul Tabelei de Procese (*Process Table*). Acesta afişează o lisă actualizată în timp real cu toate procesele care rulează:

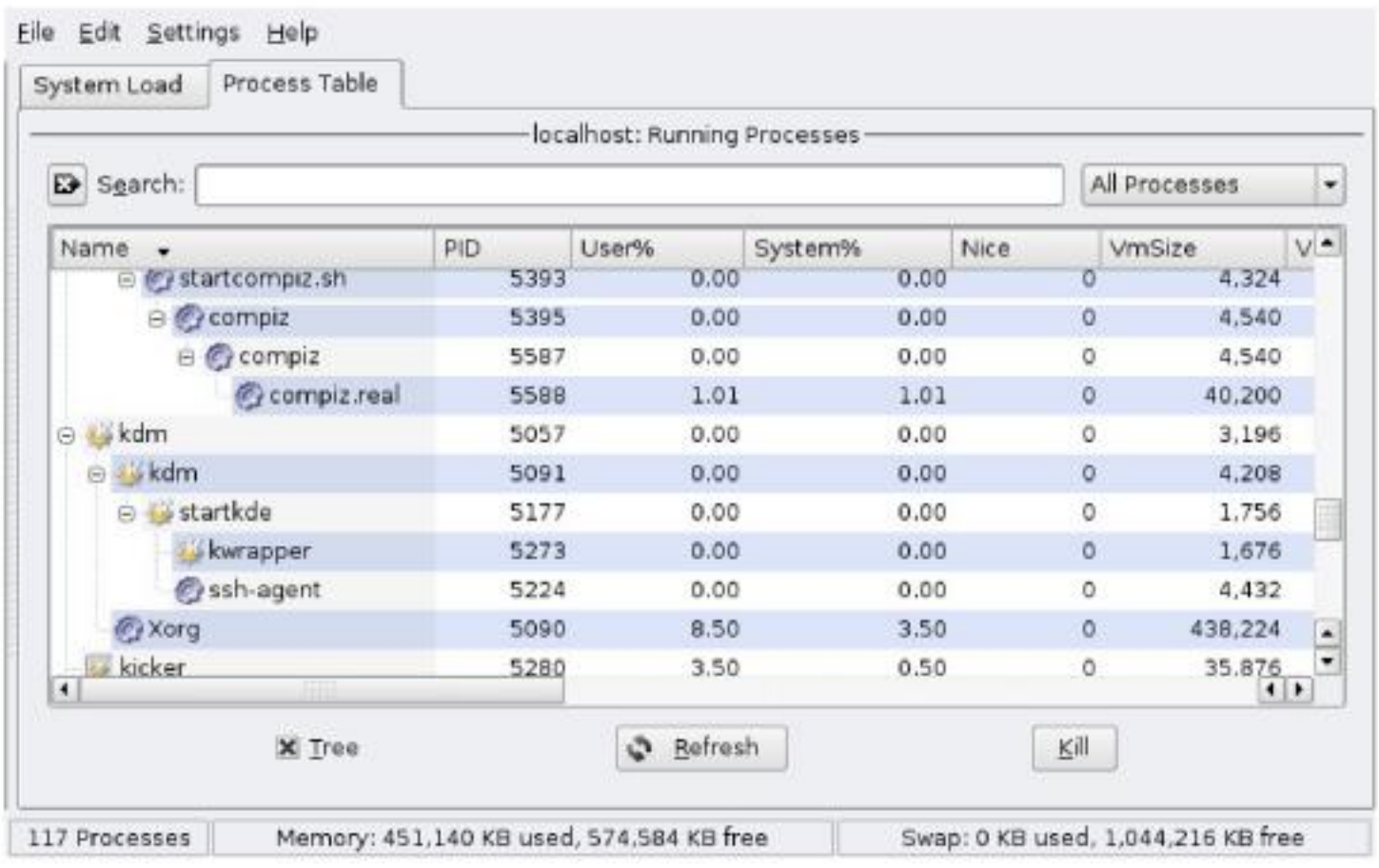


Figura 12-18 Process Table

Lista de procese poate fi afişată sub formă de arbore în cazul în care este bifată opţiunea *Tree*. De asemenea, în partea de sus a ferestrei se află un câmp *Search* se permite căutarea unui proces din listă după numele lui.

Un proces poate fi oprit în mod forţat dacă este selectat din listă şi este apăsat butonul *Kill*.

12.5 Gestiunea pachetelor

Pentru a realiza gestiunea pachetelor folosind interfaţa grafică există mai multe utilitare. Unul dintre ele este Adept Manager:

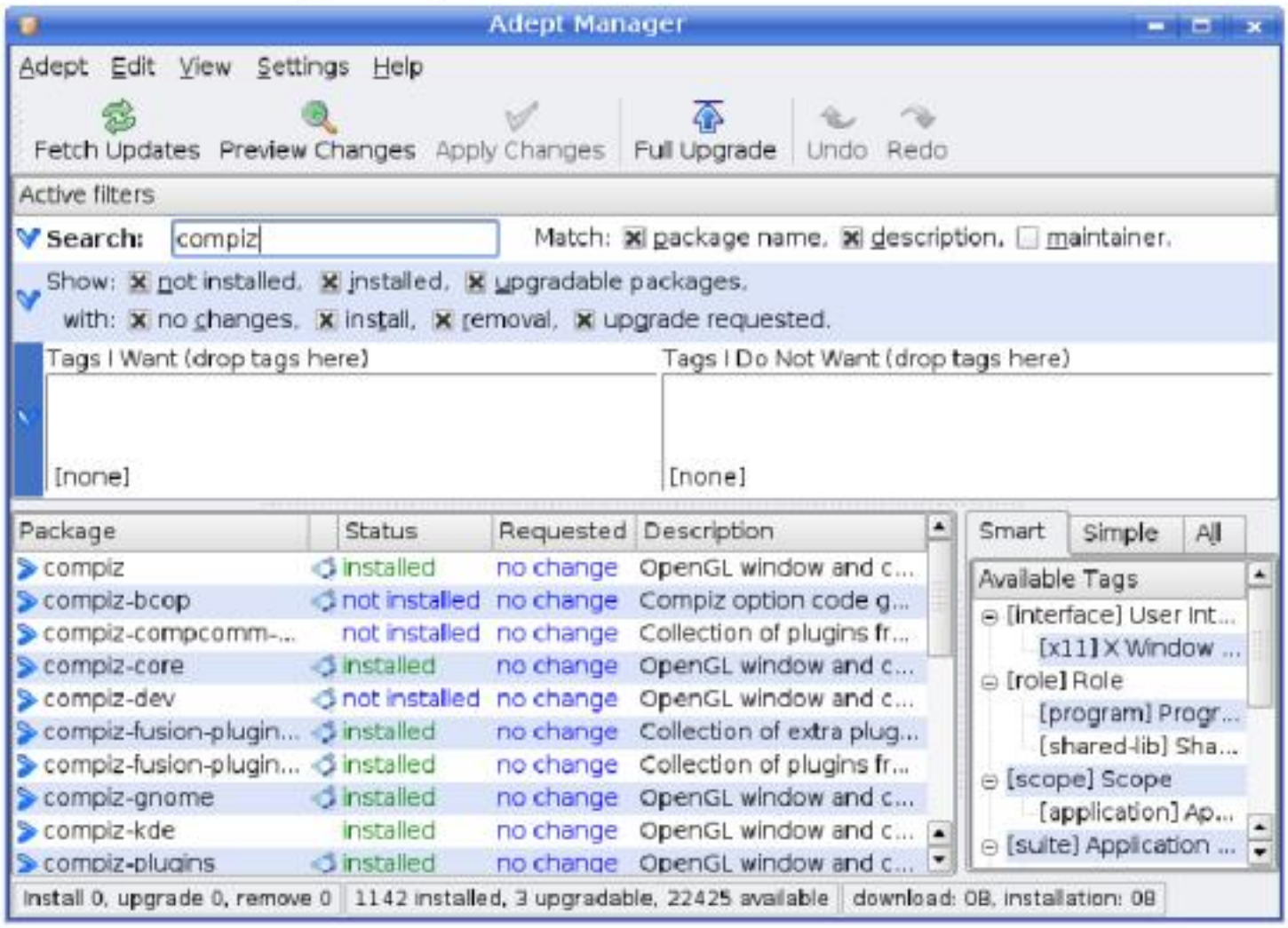


Figura 12-19 Adept Manager

El permite gestiunea repository-urilor, actualizarea bazei de date cu pachete, căutarea de pachete după nume, instalarea de pachete noi, dezinstalarea de pachete sau actualizarea tuturor pachetelor din sistem.

12.6 Aplicaţii KDE vs GNOME

Alegerea între KDE şi GNOME aparţine fiecărui utilizator şi se bazează pe mai multe criterii plecând de la aspectul vizual la cantitatea de resurse consumate. Cum este greu să fie prezentate atât aplicaţii specifice KDE cât şi aplicaţii specifice GNOME, în continuare vor fi menţionate programele implicite din KDE şi GNOME utilizate pentru aplicaţii uzuale:

Aplicație	KDE	GNOME
Configurarea Sistemului	System Setings	System | Settings
Gestiunea pachetelor	Adept	Synaptic Packet Manger
Monitorizarea sistemului	KSysGuard	System Monitor
File Manager	Konqueror, Dolphin	Nautilus
Editor de text	Kate	Gedit
Emulator de consolă	Konsole	gnome-terminal
Calculator	Kcalc	Gcalctool
Video Player	Kaffeine	Totem
Audio Player	Amarok	Rhythmbox
Înscriptor de CD/DVD	K3B	Brasero
Vizualizator de imagini	KView	Eye of GNOME
Browser Web	Konqueror	Epiphany

Aplicațiile prezentate în tabel nu reprezintă restricții. Cele mai multe dintre ele por rula atât pe şi cât şi pe GNOME dacă sunt instalate bibliotecile de care au nevoie (de exemplu GTK-ul şi Glib-ul trebuie instalate pentru a putea rula aplicații Gnome pe KDE). Singurele restricții în acest sens sunt aplicațiile de administrare a sistemului de operare.

12.7 Jocuri

Datorită originilor şi mediului căruia i-a fost destinat până nu demult, Linuxul nu a reprezentat un segment de piață atractiv pentru producătorii de jocuri. De aceea nu există foarte multe jocuri care pot rula pe acest sistem de operare.

În Windows, cele mai multe jocuri folosesc pentru desenarea obiectelor 3D API-ul Direct3D de la Microsoft, parte a API-ului multimedia DirectX (DirectX a ajuns la versiunea 10 odată cu apariția Windows Vista). Pe lângă acest API (proprietar Microsoft), există şi jocuri care folosesc un alt API, open source, numit OpenGL (*Open Graphics Library*). OpenGL este o colecție de funcții gândite a fi *cross-language* şi *cross-platform*. Este printre puținele API-uri standardizate folosite în Linux pentru crearea obiectelor 3D şi implicit a jocurilor 3D.

Cele mai multe dintre jocurile disponibile în Linux sunt 2D. Chiar dacă acest lucru nu este atractiv în ziua de azi, ele au o abordare diferită față de jocurile similare din Windows, necesitând de multe ori gândire şi nu dexteritate. Printre jocurile 2D din Linux se numără Rocks 'n' Diamonds şi KGoldrunner.

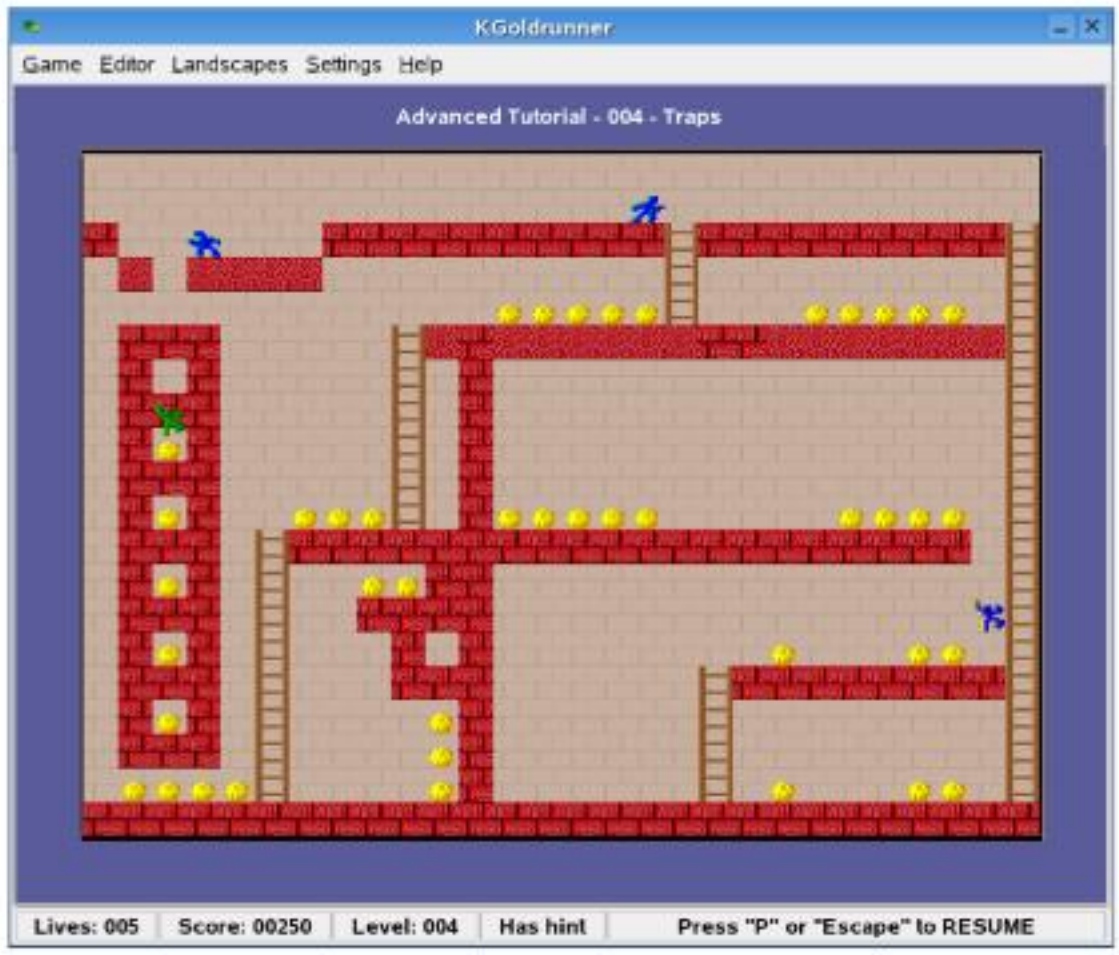


Figura 12-20 Captură de ecran din KGoldrunner

Deşi mai puţine ca număr decât jocurile 2D, jocurile 3D din Linux sunt asemănătoare ca performanţe cu cele din Windows. Pe Linux există atât *remake*-uri (ca de exemplu FreeCiv, *remake* al CivilisationII) cât şi jocuri originale (cum ar fi Alien Arena 2007, joc ce foloseşte motorul grafic QuakeII, sau Nexuiz).

Figura 12-21 Captură de ecran din Nexuiz

12.8 Studii de caz

12.8.1 Compiz. Beryl. Compiz Fusion

Introducere

Unul dintre cele mai spectaculoase elemente grafice disponibile în Linux în momentul de față este interfața grafică 3D oferită de Compiz sau Beryl. Aceasta oferă, printre altele, posibilitatea plasării desktop-urilor pe fetele unui cub şi schimbarea desktop-ului prin rotirea cubului.

Atât **Compiz** cât şi **Beryl** sunt două *window managere* mai complexe, denumite *composing window managers*, ce folosesc pentru crearea imaginii finale a desktop-ului tehnica numită compoziție (*composing*). Utilizată în special în televiziune şi cinematografie, compoziția reprezintă combinarea elementelor vizuale din mai multe surse pentru a crea o singură imagine, de cele mai multe ori cu scopul de a crea impresia că toate elementele fac parte din aceeaşi scenă. În cazul *window managerelor* aceasta înseamnă că fiecare fereastră este desenată pe un strat (*layer*) separat şi la final toate straturile sunt suprapuse pentru a crea imaginea finală, luând în considerare şi transparențele din fiecare strat.

Composing window managerele existente în momentul de față sunt Beryl, Compiz, Compiz Fusion, Metacity, Xfwm şi xcompmgr.

Compiz este unul dintre primele *composing window managere* care creează compoziția folosind hardware-ul disponibil în placa video şi nu procesorul. Acesta implementează standarde care îi permit să înlocuiască *window managerul* Metacity din GNOME sau KWin din KDE.

Prima versiune de Compiz a apărut în ianuarie 2006 şi a fost creată de Novell (dezvoltatorul SUSE). Inițial Compiz putea rula doar peste o implementare particulară a serverului X numită **Xgl** care realiza compoziția imaginilor folosind **OpenGL,** utilizând hardware-ul pus la dispoziție de placa video. Din mai 2006 acesta poate rula şi peste X.org, una din metodele de interfațare cu OpenGL fiind **AIGLX** (*Accelerated Indirect GLX*). AIGLX foloseşte şi el hardware-ul pus la dispoziție de placa video pentru a realiza compoziția imaginilor.

Tot în 2006 a început dezvoltarea, în paralel cu Compiz, a unei ramuri a proiectului numită Quinnstorm. În septembrie 2006 diferențele dintre Compiz şi Quinnstorm au devenit atât de mari încât a fost considerată imposibilă integrarea unuia în celălalt. Prin urmare Quinnstorm a fost anunțat ca un proiect independent, numit **Beryl**. Dezvoltarea lui a continuat în paralel cu Compiz.

În martie 2007 dezvoltatorii celor două proiecte au hotărât să le unească, astfel încât Beryl a fost reintegrat în Compiz. Noul proiect format în acest fel se numeşte **Compiz Fusion**. Practic Compiz Fusion este o colecție de plugin-uri, în mare parte derivate din Beryl, care funcționează împreună cu Compiz.

Instalarea Compiz Fusion

În momentul de față sunt disponibile pentru instalare toate cele trei *composing window managere*: Compiz, Beryl şi Compiz Fusion. Dintre ele, Beryl nu mai este dezvoltat, iar Compiz Fusion este încă la început şi în momentul de față nu garantează stabilitate. Totuşi Compiz Fusion este soluția care va fi utilizată de acum înainte, şi de aceea vor fi prezentați în continuare paşii necesari instalării lui.

Pentru instalare există multe tutoriale disponibile pe Internet, cele mai multe dintre ele pe forumurile diferitelor distribuții. Pentru Ubuntu/Kubuntu pachetele necesare instalării Compiz Fusion se pot găsi în repository-urile universe şi multiverse (pentru versiunea Gutsy) sau în repository-uri terțe, cum ar fi `download.tuxfamily.org/3v1deb`. Pentru a putea fi instalate pachete din aceste repository-uri, ele trebuie configurate în fişierul `/etc/apt/sources.lst`.

Pentru configurarea repository-urile universe şi multiverse liniile din `/etc/apt/sources.lst` sunt următoarele:

```
deb http://ro.archive.ubuntu.com/ubuntu/ feisty universe multiverse
deb-src http://ro.archive.ubuntu.com/ubuntu/ feisty universe multiverse
```

Pentru configurarea unui reposotiry terţ (de exemplu cel de la `download.tuxfamily.org/3v1deb`), liniile din `/etc/apt/sources.lst` sunt următoarele:

```
deb http://download.tuxfamily.org/3v1deb feisty eyecandy
deb-src http://download.tuxfamily.org/3v1deb feisty eyecandy
```

După ce a fost modificat fişierul `/etc/apt/sources.lst`, trebuie actualizată baza de date cu pachete disponibile cu ajutorul comenzii `apt-get update`. Apoi trebuiesc instalate pachetele următoare pentru KDE:

```
asgard:/home/george# apt-get install compiz compizconfig-settings-manager
compiz-fusion-plugins-extra compiz-kde libcompizconfig-backend-kconfig
```

sau pentru GNOME:

```
asgard:/home/george# apt-get install compiz compizconfig-settings-manager
compiz-fusion-plugins-extra compiz-gtk libcompizconfig-backend-gconf
```

În momentul acesta Compiz ar trebui să fie instalat. Pentru a îl porni re rulează comanda:

```
asgard:/home/george$ compiz --replace
```

O problemă des întâlnită în acest moment este blocarea *window managerului*. În acest caz ferestrele nu mai au bară de titlu, nu mai pot fi închise şi nici mutate.

În cazul în care Compiz a fost instalat cu succes, prin apăsarea simultană a tastelor Ctrl+Alt+Butonul stânga al mouse-ului şi prin mutarea mouse-ului se va putea roti cubul pe care sunt amplasate desktop-urile:

Figura 12-22 Amplasarea desktop-urilor pe feţele unui cub

Configurarea Compiz

Pentru configurarea Compiz se foloseşte *CompizConfig Settings Manager*. Acesta poate fi accesat, în cazul KDE, din meniul Start | Settings | CompizConfig Settings Manager.

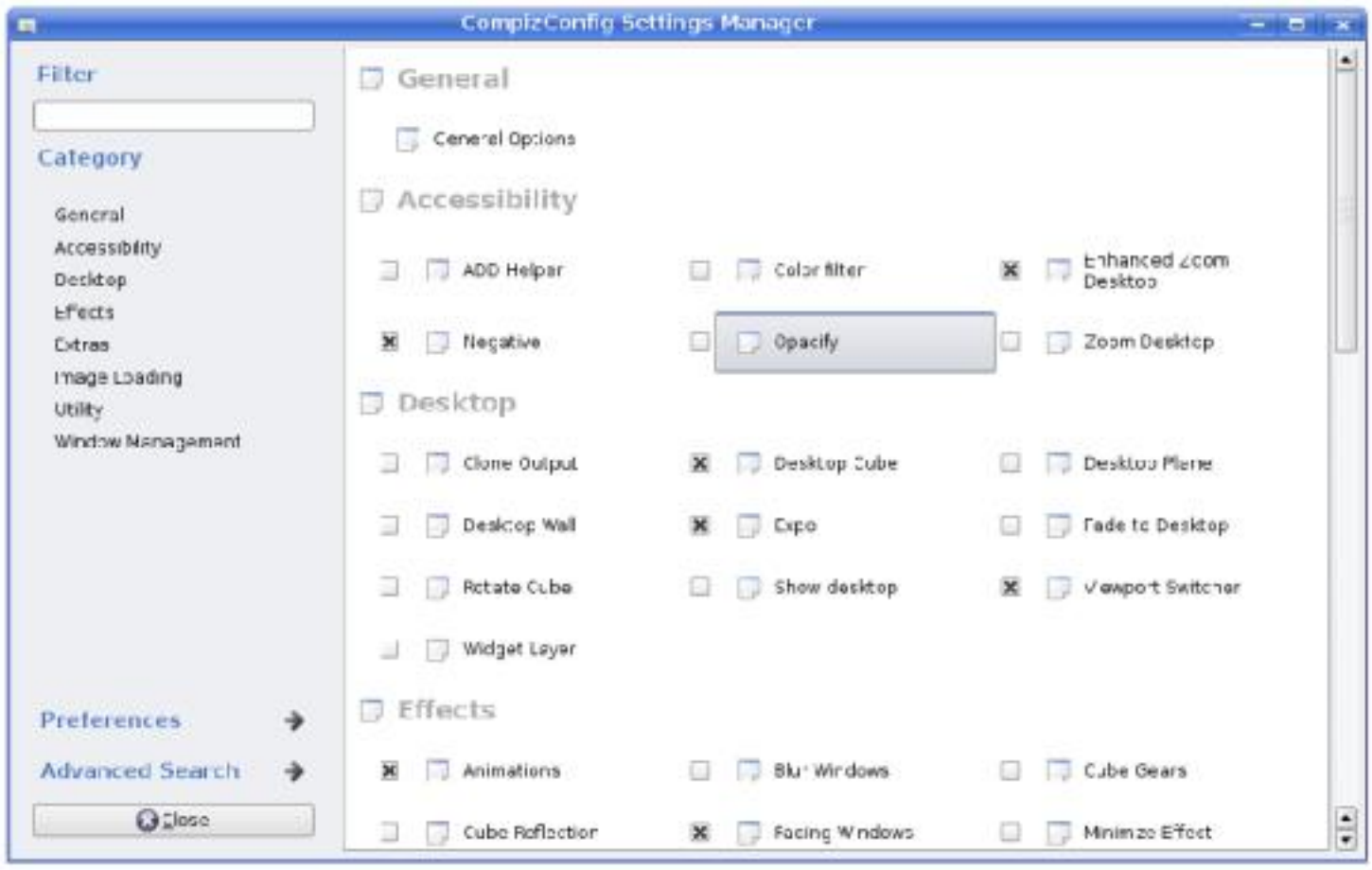


Figura 12-23 CompizConfig Settings Manager

Din această interfață se pot activa/dezactiva şi configura toate plugin-uile Compiz-ului. Spre exemplu conținutul desktop-urilor poate fi vizualizat sub forma unui film foto, trecerea între desktop-uri făcându-se foarte rapid:

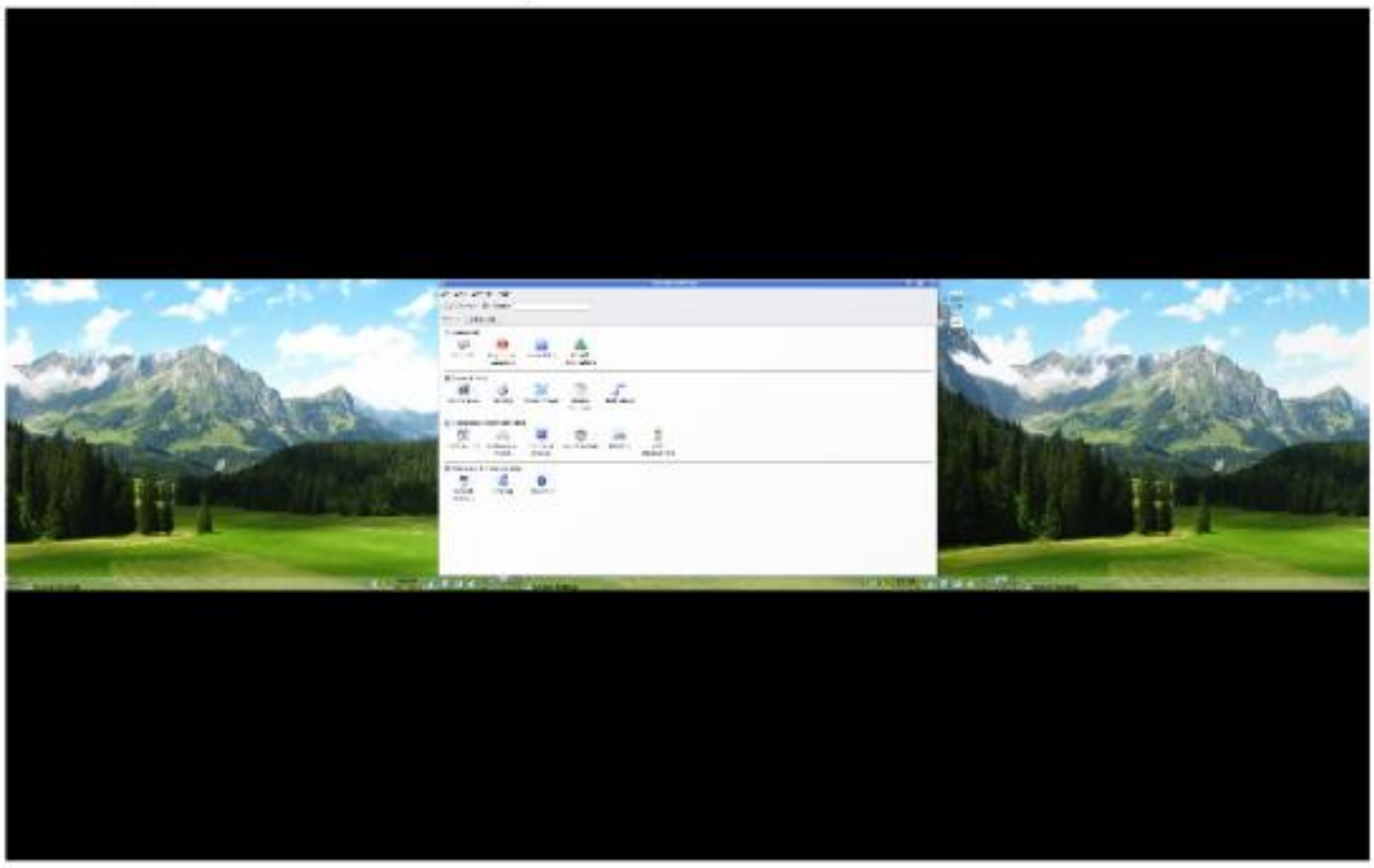


Figura 12-24 Vizualizarea conținutului tuturor desktop-urilor

Un alt plugin este cel care permite deformare elastică a ferestrelor atunci când sunt mutate (*Wobbly Windows*):

Figura 12-25 Wobbly Windows

Un alt plugin spectaculos este *Paint fire on screen*. Acesta permit desenarea pe ecran, cu mouse-ul, folosind foc pe post de cerneală:

Figura 12-26 Paint fire on the screen

12.8.2 Configurarea X peste SSH

Aşa cum a fost prezentat şi la începutul capitolului, arhitectura *X Window System* permite comunicarea prin reţea între clienţii X şi serverul X. Această comunicare se poate realiza folosind mai multe protocoale, printre care şi SSH:

SSH [vezi 8.2.2] este un protocol care permite schimbul de informaţie între două calculatoare utilizând un canal securizat. SSH poate încapsula date ce reprezintă comenzi, rezultatele comenzilor, fişiere, etc. Printre datele ce pot fi încapsulate se numără şi informaţiile schimbate între clienţii X şi serverul X, în acest mod realizându-se transmiterea protocolului X peste protocolul SSH (*X11 Forwarding*).

Pentru a putea rula un client X aflat la distanţă peste SSH primul pas este configurarea protocolului SSH în acest sens. Cea mai simplă metodă este specificarea parametrului `-x` atunci când se realizează conexiunea ssh de pe calculatorul ce interacţionează cu utilizatorul pe calculatorul pe care va rula clientul X.

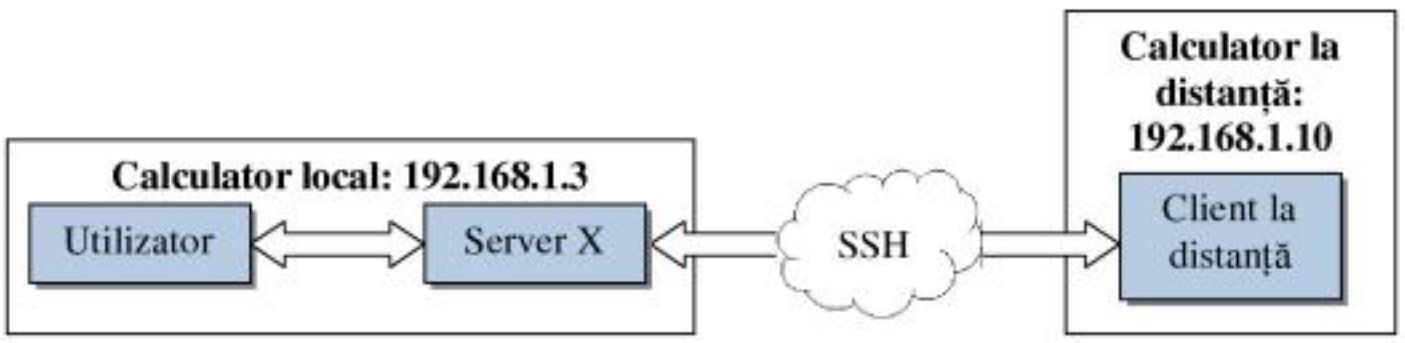


Figure 12-1 Conectarea între client X şi un server X peste SSH

Comanda cu care se realizează conexiunea SSH pentru a permite încapsularea protocolului X este:

```
george@asgard:~$ ssh george@192.168.1.10 -X
```

După ce conexiunea SSH a fost stabilită se poate porni un client X. De exemplu se poate porni KCalc:

```
george@acasa:~$ kcalc
```

În urma acestei comenzi va apărea fereastra programului KCalc:

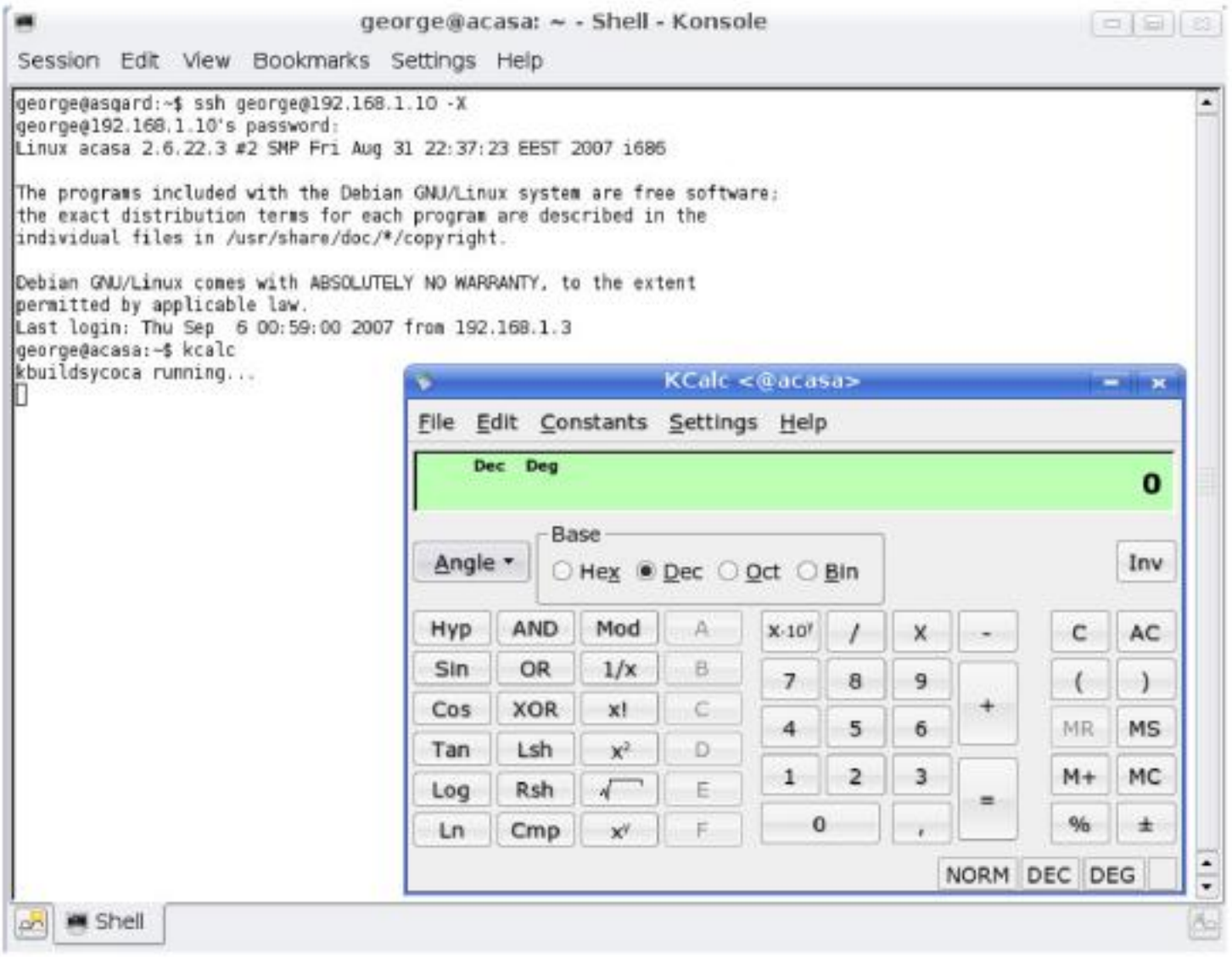


Figura 12-27 Pornirea unei aplicații X prin ssh

După cum se poate observa bara de titlu a ferestrei KCalc specifică faptul că această fereastră corespunde unui client care nu se găseşte pe calculatorul local (`<@acasa>`).

Folosind X peste SSH este posibil să fie rulate nu doar programe individuale, ci şi un întreg *Desktop Environment*. Acest lucru nu este însă uzual datorită alternativelor mai bune existente în acest sens (vnc, freenx, etc). Conexiunea SSH, folosind un canal de comunicație securizat, necesită criptare şi decriptare. Acestea necesită trimiterea prin rețea a unei cantități suplimentare de informații şi consumă resurse hardware pentru a putea realiza criptarea şi decriptarea.

Pentru a putea rula X peste SSH folosind pe calculatorul local Windows şi pe cel aflat la distanță Linux trebuie ca pe calculatorul local să fie instalat Cygwin.

12.8.3 Instalarea şi configurarea VNC

VNC (*Virtual Network Computing*) este unul dintre cele mai folosite sisteme de vizualizat desktop-ul unui calculator aflat la distanță. VNC se bazează pe protocolul RFB (*Remote Frame Bufer*) şi de aceea poate fi folosit între sisteme de operare diferite (Windows, Linux, MacOS).

VNC funcționează pe sistemul client – server. Pe calculatorul aflat la distanță (al cărui desktop va fi vizualizat) rulează un server de VNC la care se conectează un client aflat pe calculatorul local. La un server se pot conecta mai mulți clienți simultan. Un client poate doar să vizualizeze desktop-ul calculatorului pe care rulează serverul sau poate să şi interacționeze cu el (de exemplu schimbând poziția cursorului mouse-ului).

Figura 12-28 Conectarea prin VNC folosind un client Windows şi un server Linux

Serverul de VNC preia de câteva ori pe secundă imaginea care se afişează pe ecran şi o trimite tuturor clienţilor. În cazul în care clienţii îi cer acest lucru, serverul poate accepta de la aceştia cereri de modificare a cursorului mouse-ului de efectuare de click-uri de mouse sau de introducere de caractere. În acest fel utilizatorul ce interacţionează cu calculatorul pe care rulează clientul de VNC poate prelua controlul calculatorului pe care rulează serverul de VNC.

Folosind VNC este posibil ca mai mulţi utilizatori să aibă simultan controlul unui calculator.

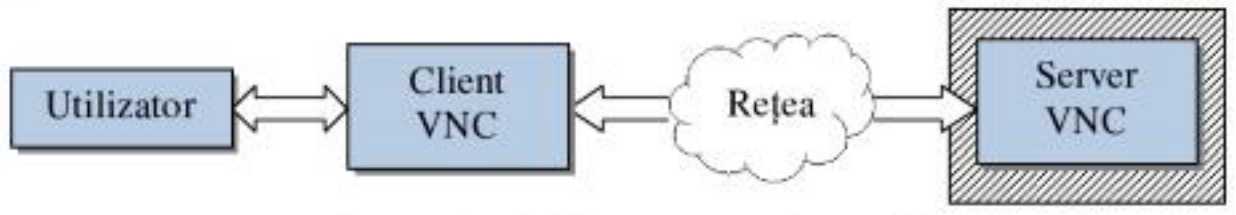


Figura 12-29 Conectarea prin VNC

De cele mai multe înainte de a putea accesa un server de VNC utilizatorul trebuie să se autentifice. Acest lucru se realizează pe baza unui nume de utilizator şi a unei parole.

Atât în Windows cât şi în Linux există mai multe programe, atât server cât şi client, care implementează VNC.

Pentru Linux, unul dintre cele mai simple servere VNC este `x11vnc`. Odată instalat, acest server este pornit prin apelarea comenzii `x11vnc`:

```
george@acasa:~$ x11vnc -usepw
10/09/2007 22:48:59 -usepw: found /home/george/.vnc/passwd

[...]
10/09/2007 22:48:59 *** Trying ":0" in 4 seconds.  Press Ctrl-C to abort.
10/09/2007 22:48:59 *** 1 2 3 4
10/09/2007 22:49:03 *** XOpenDisplay of ":0" successful.
10/09/2007 22:49:03
10/09/2007 22:49:03 Using X display :0
10/09/2007 22:49:03
10/09/2007 22:49:03 X display :0.0 is 32bpp depth=24 true color

FrameBuffer Info:
```

```
 width:            1280
 height:           1024
 scaled_width:     1280
 scaled_height:    1024

[...]

The VNC desktop is:      acasa:0
PORT=5900
```

Comanda trebuie apelată de către utilizatorul care este autentificat în interfața grafică. Implicit, după ce ultimul client se deconectează, serverul se va opri.

Pentru a securiza accesul clienților pe baza unei parole serverul x11vnc trebuie pornit cu opțiunea `-usepw`. În acest caz serverul va verifica dacă există fișierul `~/.vnc/passwd`. În cazul în care fișierul există, din el va fi citită parola ce va fi cerută clienților. Dacă fișierul nu există, va fi cerută utilizatorului o parolă care va fi stocată în `~/.vnc/passwd` și va fi folosită la următoarele instanțe ale serverului.

Există mai mulți clienți VNC în Linux: `xvnc4viewer`, `xvncviewer`, `xtightvncviewer`, etc.

Și în Windows există atât servere cât clienți VNC. Unele programe sunt gratuite (ca de exemplu tightVNC), altele trebuie cumpărate (ca de exemplu RealVNC, ce dispune și de o versiune gratuită dar limitată ca opțiuni).

O alternativă la VNC este **FreeNX**. Acest program trimite datele între client și server peste SSH, însă comprimă datele înainte de a le trimite. În acest fel, bazându-se și pe un sistem de *cache*, reușește să obțină performanțe superioare VNC-ului ca timp de răspuns și lățime de bandă consumată.

12.8.4 Remote Desktop Connection

RDC (*Remote Desktop Connection*) este o aplicație care funcționează în mod similar cu VNC și care permite unui utilizator să acceseze de la distanță desktop-ul unui calculator. Această aplicație este disponibilă doar împreună cu Windows XP Professional sau cu Windows Vista Business, Ultimate sau Enterprise.

În comparație cu VNC, RDC are avantajul că este instalat o dată cu instalarea Windows-ului. Nemaifiind nevoie de un alt software pentru funcționarea sa. Dezavantajul RDC este acela că nu permite accesarea desktop-ului simultan local și de la distanță. Cu alte cuvinte, atunci când un utilizator este autentificat local pe calculator și interacționează direct cu sistemul de operare, nu este posibil să altcineva să fie autentificat prin RDC și să vadă acțiunile primului utilizator. În momentul în care se încearcă autentificarea prin RDC utilizatorul local este deconectat. Atunci când un utilizator aflat la distanță este autentificat prin RDC și un utilizator local încearcă și el să se autentifice, utilizatorul aflat la distanță este deconectat.

Pentru utilizarea RDC acesta trebuie activat pe calculatorul „server" (calculatorul care va fi controlat de la distanță). Pentru această se accesează *System Properties* (click dreapta pe My Computer, apoi Properties), după care din tabul *Remote* se bifează *Allow users to connect remotely to this computer*.

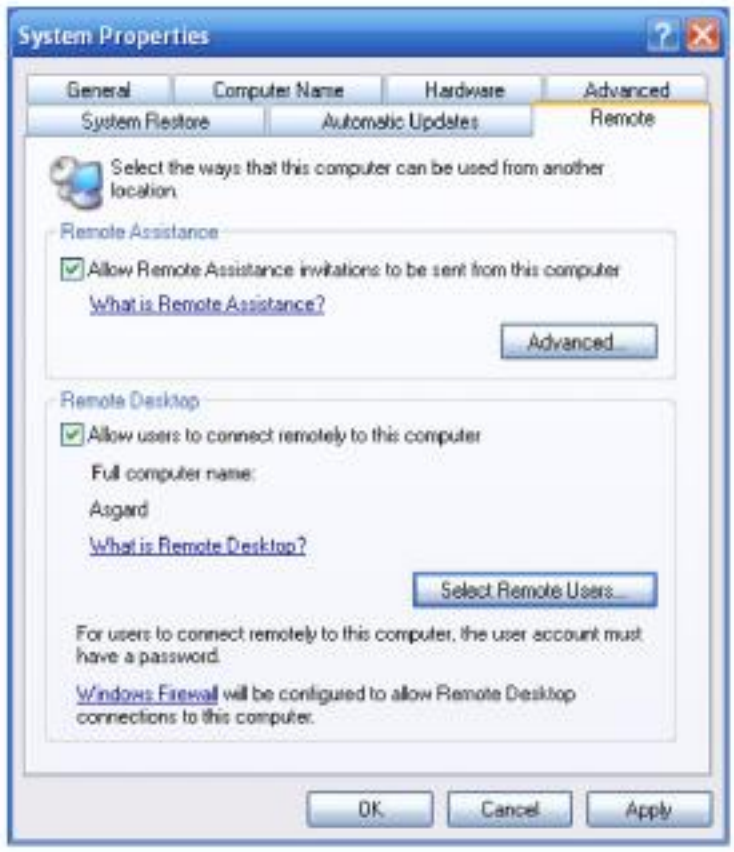


Figura 12-30 Activarea RDC

Funcţie de programele de tip *firewall* şi antivirus existente pe cele două calculatoare între care se va realiza legătura este posibil să fie necesare configurări ale *firewall-ului* sau antivirusului.

Pentru a realiza conexiunea, de pe calculatorul local trebuie pornit clientul de RDC. Acesta se accesează prin *Start* | *All Programs* | *Accessories* | *Communications* | *Remote Desktop Connection* sau *Start* | *All Programs* | *Accessories* | *Remote Desktop Connection*.

Figura 12-31 Fereastra clientului de RDC

Pentru conectarea la calculatorul aflat la distanţă este necesară adresa acestuia IP. Se pot configura parametrii conexiunii RDC, pentru aceasta trebuind accesat butonul *Options*:

Figura 12-32 Configurarea opțiunilor conexiunii RDC

După ce conexiunea a fost inițializată utilizatorul trebuie să se autentifice:

Figura 12-33 Autentificarea RDC

După ce autentificarea a fost încheiată cu succes desktop-ul calculatorului afla la distanță poate fi accesat.

Cuvinte cheie

- CLI, TUI, GUI
- X Window System
- Protocolul X
- grafică raster
- grafică vectorială
- client X
- server X
- window manager: Kwin, Metacity, Compiz
- desktop environment: KDE, GNOME
- display manager: gdm, xdm, kdm
- XFree86, X.Org
- xinit, startx
- /etc/X11/xorg.conf
- aplicații grafice
- Compiz Fusion
- composing window manager
- Xgl
- OpenGL
- AIGLX
- X Forwarding
- VNC
- tightVNC, RealVNC
- FreeNX
- RDC

Întrebări

1. *Utilizatorul Dorel încearcă să configureze interfața grafică din Linux folosind fișierul /etc/X11/xorg.conf. El modifică una din secțiunile Display care arată acum ca în exemplul de mai jos. La repornirea serverului X acesta semnalează o eroare în fișierul de configurare. Care ar putea fi cauza?*

```
SubSection "Display"
    Depth       24
    Modes       "1280.4x800.3" "1280x768" "1024x768" "800x600"
EndSubSection
```

- ❑ Adâncimea de culoare specificată în secțiunea Display modificată este invalidă
- ❑ Nu trebuiesc folosite ghilimele la începutul secțiunii Display
- ❑ Rezoluțiile definite sunt invalide
- ❑ Sunt definite prea multe rezoluții

2. *Un Display Manager are rolul de a:*

- ❑ Controla felul în care arată ferestrele
- ❑ Permite unui urilizator să se autentifice în interfața grafcă
- ❑ Controla deschiderea şi închiderea ferestrelor
- ❑ Gestiona conținutul ferestrelor

3. *Care dintre următoarele perechi nu conține elemente cu același rol?*

- ❑ KDE şi GNOME
- ❑ kdm şi gdm
- ❑ kwin şi metacity
- ❑ compiz şi xorg

4. *Utilizatorul Dorel lucrază pe un calculator pe care interfața grafică este instalată, dar nu este pornită. El încearcă să pornească interfața cu comanda „dorel@home:~$ /etc/init.d/kdm start " dar nu reuşeşte. De ce?:*

- ❑ Dorel nu are drepturi de execuție în directorul unde se află scriptul „kdm"
- ❑ Scriptul kdm nu porneste interfața grafică
- ❑ Dorel trebuia să ruleze scriptul „startx" înainte de a rula scriptul „kdm"
- ❑ Dorel trebuia să ruleze scriptul „startx" după ce a rulat scriptul „kdm"

5. *Fișierul de configurare a serverului X.org este:*
- /etc/xorg.conf
- /etc/xorg.text
- -etc/X11/x.conf
- /etc/X11/xorg.conf

6. *Care dintre urmatoarele nu este o secțiune a fișierului de configurare a serverului X?*
- InputDevice
- Refresh
- Monitor
- Screen

7. *Care dintre următoarele nu este un Window Manager?*
- AIGLX
- Compiz
- Kwin
- Metacity

8. *Care dintre următoarele aplicații este utilizată în mediul grafic pentru monitorizarea ocupării resurselor hardware?*
- Top
- KSysGuard
- Htop
- Free

9. *Un Window Manager controlează felul în care arată ferestrele aplicațiilor. Un client X preia intrările (inputul) direct de la utilizator.*
- Adevărat, Adevărat
- Adevărat, Fals
- Fals, Adevărat
- Fals, Fals

10. *Care dintre următorii itemi nu reprezintă un client X?*
- Kcalc
- gnome-terminal
- xorg
- kdm

13 Anexe

13.1 Anexa 1: Pachete

Distribuțiile Linux folosesc noțiunea de pachet pentru a reprezenta diversele forme de aplicații care pot fi instalate pe sistemul de operare. Un pachet este, aşadar, forma instalabilă a unui program pe o distribuție Linux. Distribuțiile pot folosi un format propriu de împachetare. Cele mai cunoscute formate de pachete sunt formatul DEB folosit inițial de Debian şi RPM (*RPM Packet Manager*) folosit inițial de RedHat.

În tabelul de mai jos sunt prezentate comenzile folosite la gestiunea pachetelor pe diverse distribuții Linux:

Acțiune	Ubuntu / Debian / Knoppix	Red Hat / Fedora Core / Mandriva	SuSE	Gentoo
actualizare bază de date de pachete	`apt-get update`	`(N/A)`	(În GUI)	`emerge --sync`
upgrade sistem	`apt-get upgrade`	`yum update | yum upgrade`	(În GUI)	`emerge --update`
instalare pachet din repository	`apt-get install`	`yum install`	`yast2 -i`	`emerge`
instalare pachet din fişier	`dpkg -i`	`rpm -i | yum localinstall`	(În GUI)	`emerge <tbz2file>`
dezinstalare pachet	`apt-get remove`	`yum remove`	`yast2 --remove`	`emerge --unmerge`
căutare pachet	`apt-cache search`	`yum list`	(În GUI)	`emerge --search`
apartenență fişier X la un pachet	`dpkg -S`	`rpm -qf | yum provides`	(În GUI)	(N/A)
conținut pachet X	`dpkg -L`	`rpm -qpl`	(În GUI)	(N/A)
informații despre pachetul X	`apt-cache show`	`rpm -qpi`	(În GUI)	(N/A)

13.2 Anexa 2: Acronime

- **ACL** - Access Control List
- **ACPI** - Advanced Configuration and Power Interface
- **AGP** - Accelerated Graphics Port
- **AP** - Access Point
- **API** - Application Programming Interface

- **ASCII** - American Standard Code for Information Interchange
- **ATA** - Advanced Technology Atta
- **BASH** - Bourne Again SHell
- **BIOS** - Basic Input Output System
- **BSD** - Berkeley Software Distribution
- **CA** - Certification Authority
- **CD** - Compact Disc
- **CLI** - Command Line Interface
- **CMOS** - Complementary Metal-Oxide-Semiconductor (memorie BIOS nevolatila)
- **CPU** - Central Processing Unit
- **DDR** - Double Data Rate
- **DHCP** - Dynamic Host Configuration Protocol
- **DNS** - Domain Name System
- **DoS** - Denial of Service
- **DOS** - Disk Operating System
- **DRAM** - Dynamic Random Access Memory
- **DVD** - Digital Video Disc
- **ELF** - Executable And Linking Format
- **FAT** - File Allocation Table
- **FDD** - Floppy Disk Drive
- **FLOSS** - Free/Libre/Open Source Software
- **FTP** - File Transfer Protocol
- **GCC - GNU** Compiler Collection
- **GID** - Group ID
- **GNOME - GNU** Object Model Environment
- **GNU** - Gnu's Not Unix
- **GPL** - General Public License
- **GRUB** - GRand Unified Bootloader
- **GUI** - Graphical User Interface
- **HDD** - Hard Disk Drive
- **HTML** - HyperText Markup Language
- **HTTP** - HyperText Transfer Protocol
- **IBM** - International Business Machines
- **ICMP** - Internet Control Message Protocol
- **IDE** - Integrated Drive Electronics
- **IMAP** - Internet Message Access Protocol
- **IP** - Internet Protocol
- **KDE** - K Desktop Environment
- **LAN** - Local Area Network
- **LILO** - LInux LOader
- **LINUX** - Linus' Unix
- **MAN** - Metropolitan Area Network
- **MBR** - Master Boot Record
- **NTFS** - New Technology Filesystem
- **NTLDR** - NT Loader
- **OS** - Operating System
- **PCI** - Peripheral Component Interconnect
- **PDF** - Portable Document Format
- **PID** - process id
- **PKI** - Public Key Infrastructure
- **PnP** - Plug and Play
- **POP3** - Post Office Protocol v3
- **POST** - Power-On Self Test
- **RAM** - Random Access Memory
- **ROM** - Read Only Memory
- **RTOS** - Real Time Operating System
- **SATA** - Serial ATA
- **SCSI** - Small Computer System Interface
- **SDRAM** - Synchronous Dynamic Random Access Memory
- **SMART** - Self-Monitoring, Analysis and Reporting Technology
- **SMB** - Server Message Block
- **SMTP** - Simple Mail Transfer Protocol
- **SO** - Sistem de Operare
- **SSH** - Secure SHell
- **TCP** - Transmission Control Protocol
- **TTL** - Time To Live
- **TTY** - TeleTYpe (terminal de date)
- **TUI** - Text User Interface
- **UDP** - User Datagram Protocol
- **UID** - User ID
- **UNIX** - Uniplexed Information and Computing Service
- **UPS** - Uninterruptible Power Source
- **URL** - Uniform Resource Locator
- **USB** - Universal Serial Bus
- **UTP** - Unshielded Twisted Pair
- **VNC** - Virtual Network Computing
- **WAN** - Wide Area Network
- **WWW** - World Wide Web
- **WYSIWYG** - What You See Is What You Get

Cuprins:

1 Introducere 1
1.1 Ce este un sistem de operare? 1
1.2 Scurt istoric al sistemelor de operare 5
1.3 Sisteme Windows versus sisteme Linux? 7
1.4 Studii de caz 9
2 Instalarea Linux. Configurări de bază 12
2.1 Linux. Distribuții de Linux 12
2.2 Partiții 16
2.3 Instalare Kubuntu 17
2.4 Interacțiunea cu sistemul de operare 25
2.5 Configurări de bază ale SO 33
2.6 Studii de caz 36
3 Sistemul de fişiere 41
3.1 Noțiuni introductive 41
3.2 Lucrul cu sistemele de fişiere 49
3.3 Tipuri de fişiere 53
3.4 Operații uzuale asupra fişierelor şi directoarelor 56
3.5 Redirectări de comenzi 63
3.6 Drepturi de acces 64
3.7 Tendințe în sistemele de fişiere 67
3.8 Studii de caz 68
4 Procese 74
4.1 Noțiuni introductive 74
4.2 Ierarhia de procese în Unix. Vizualizarea proceselor sistemului 78
4.3 Rularea proceselor în background. Job-uri. Daemoni 89
4.4 Semnale 93
4.5 Comunicația între procese 97
4.6 Swapping 98
4.7 Studii de caz 99
5 Pornirea şi inițializarea sistemului 107
5.1 Pornirea sistemului 107
5.2 Bootloader 110
5.3 Încărcarea nucleului 119
5.4 Inițializarea sistemului 120
5.5 Studiu de caz 127
6 Analiza hardware a sistemului 132
6.1 Structura unui sistem de operare 132
6.2 Considerente hardware 134
6.3 Suportul pentru dispozitive la nivelul kernel-ului 137
6.4 Analiza hardware a unui sistem (magistrale, chipset, CPU, memorie, dispozitive) . 138
6.5 Interfața cu dispozivitele din userspace - udev şi /dev 142
6.6 Lucrul cu dispozitive (/dev): 143
6.7 Analiza proceselor si a kernel-ului 146
6.8 Studii de caz: 148

7 Configurări de rețea de bază ... 153
7.1 Concepte de rețea ... 153
7.2 Parametri de rețea ... 155
7.3 Configurări temporare ... 159
7.4 Configurări permanente ... 163
7.5 Testarea configurațiilor de rețea ... 166
7.6 Studii de caz ... 170
8 Servicii de rețea ... 182
8.1 Concepte specifice aplicațiilor de rețea ... 182
8.2 Execuția comenzilor la distanță ... 188
8.3 Email - Poşta electronică ... 194
8.4 WWW ... 201
8.5 Partajarea de fişiere ... 206
8.6 Studiu de caz: ... 213
9 Elemente de securitate şi monitorizare ... 219
9.1 Problematica securității ... 219
9.2 Securizarea sistemului ... 220
9.3 Întreținerea sistemului ... 229
9.4 Riscuri de securitate în rețele de calculatoare ... 234
9.5 Monitorizarea de rețea ... 237
9.6 Studiu de caz ... 242
10 Compilare, linking şi execuție ... 249
10.1 Introducere ... 249
10.2 Compilare. GCC ... 252
10.3 Biblioteci de funcții ... 262
10.4 Automatizarea sarcinilor – make ... 264
10.5 Portabilitate ... 266
10.6 Execuția programelor ... 268
10.7 Studiu de caz ... 270
11 Shell scripting ... 279
11.1 Noțiuni introductive ... 279
11.2 Interacțiunea cu shell-ul ... 281
11.3 Scripturi shell ... 286
11.4 Programarea shell ... 295
11.5 Filtre de text ... 309
11.6 Comenzi de lucru cu fişiere ... 317
11.7 Expandarea în shell ... 321
11.8 Parametrii unui script shell ... 323
11.9 Funcții ... 325
11.10 Studii de caz ... 329
12 Mediul grafic ... 336
12.1 Interfața grafică în Linux. Componente ... 336
12.2 Pornirea şi oprirea interfeței grafice ... 342
12.3 Configurarea serverului X ... 343
12.4 Configurarea sistemului din KDE ... 345
12.5 Gestiunea pachetelor ... 353
12.6 Aplicații KDE vs GNOME ... 353
12.7 Jocuri ... 354
12.8 Studii de caz ... 356
13 Anexe ... 368
13.1 Anexa 1: Pachete ... 368
13.2 Anexa 2: Acronime ... 368

Bibliografie:

- Acostachioaie, D.: Administrarea si configurarea sistemelor Linux (Editia a II-a), Iasi: Polirom 2003
- Acostachioaie, D.: Securitatea sistemelor Linux, Iasi: Polirom 2003
- Barrett, Daniel. Linux Pocket Guide. Sebastopol: O'Reilly, 2004.
- Bellomo, M.: Debian GNU/Linux for Dummies, New York: Hungry Minds, 2000
- Benvenuti, Christian. Understanding Linux Network Internals. Sebastopol: O'Reilly Media, Inc, 2005
- Bovet, Daniel and Marco Cesati. Understanding the Linux Kernel. City: Oreilly & Associates Inc, 2005.
- Comer, D.E., Droms, R.E.: Computer Networks and Internets, with Internet Applications (3rd Edition), Upper Saddle River, NJ: Prentice Hall, 2003
- Corbet, Jonathan et.al. Linux Device Drivers. Sebastopol: O'Reilly, 2005.
- Dalheimer, Matthias and Matt Welsh. Running Linux. Sebastopol: O'Reilly Media, Inc, 2005.
- Davis, William. Operating Systems : a Systematic View. Harlow: Addison Wesley, 2004.
- Deitel, Harvey. Operating Systems. Reading: Addison-Wesley Pub. Co, 2004.
- Eckert, Jason and M. John Schitka. Linux+ Guide to Linux Certification, Second Edition. Cambridge: Course Technology, 2005.
- Eckert, Jason. Getting Started with Linux: Novell's Guide to Comptia's Linux+. Cambridge: Course Technology, 2006.
- Farrel, A.: The Internet and Its Protocols, San Francisco: Morgan Kafmann, 2004
- Flickenger, Rob. Linux Server Hacks. Sebastopol: O'Reilly, 2003.
- Fox, Tammy. Red Hat Enterprise Linux 5 Administration Unleashed. Indianapolis: SAMS, 2007.
- Fusco, John. The Linux Programmer's Toolbox. Upper Saddle River: Prentice Hall PTR, 2007.
- Hall, E.: Internet Application Protocols: The Definitive Guide, San Francisco, CA: O'Reilly, 2003
- Held, G.: Understanding Data Communications: From Fundamentals to Networking, 3rd Edition, New York: John Wiley & Sons, 2000
- Hennessy, John and David Patterson. Computer Architecture, Fourth Edition. San Diego: Morgan Kaufmann, 2006.
- Hunger, S.: Debian GNU/Linux Bible, New York: John Wiley & Sons, 2001
- Hunt, C.: TCP/IP Network Administration (3rd Edition), San Francisco, CA: O'Reilly, 2002
- Jones, M. Gnu/Linux Application Programming. London: Charles River Media, 2005.
- Karris, S.T.: Networks: Design and Management, Fremont, CA: Orchard Publications, 2002
- Knuth, Donald. The Art of Computer Programming. Boston: Addison-Wesley, 1997.
- Kurose, J.F., Ross, K.W.: Computer Networking : A Top-Down Approach Featuring the Internet (3rd Edition), Boston: Addison Wesley, 2004

- Liotine, M.: Mission-Critical Network Planning, Norwood, MA: Artech House Publishers, 2003
- Love, Robert. Linux Kernel Development. San Jose: Novell Press, 2005.
- Love, Robert. Linux System Programming: Talking Directly to the Kernel and C Library. Sebastopol: O'Reilly Media, Inc, 2007.
- Mann, s.: LINUX TCP/IP Network Administration, Upper Saddle River, NJ: Prentice Hall, 2001
- Matthew, Neil and Richard Stones. Beginning Linux Programming. Indianapolis: Wrox, 2007.
- McCabe, J.D.: Network Analysis, Architecture and Design, Second Edition, San Francisco: Morgan Kaufmann, 2003
- Negus, Christopher and Chris Negus. Linux Bible 2005 Edition. New York: Wiley, 2005.
- Nemeth, Evi et.al. Linux Administration Handbook. Upper Saddle River: Prentice Hall PTR, 2006
- Palmer, Michael. Guide to Unix Using Linux, Third Edition. Cambridge: Course Technology, 2004
- Peterson, L.L., Davie, B.S.: Computer Networks: A Systems Approach, 3rd Edition, San Francisco: Morgan Kafmann, 2003
- Rughiniş, R., Deaconescu, R.: Administrarea reţelelor locale, Printech, 2006
- Schroder, Carla. Linux Cookbook. Sebastopol: O'Reilly, 2004.
- Shah, Steve and Wale Soyinka. Linux Administration. City: McGraw-Hill Osborne Media, 2005.
- Shingledecker, Robert et.al. The Official Damn Small Linux. Upper Saddle River: Prentice Hall PTR, 2007.
- Siever, Ellen et.al. Linux in a Nutshell. Sebastopol: O'Reilly, 2005.
- Silberschatz, Abraham et.al: Operating System Concepts. Chichester: John Wiley & Sons, 2004.
- Smith, Bob et.al. Linux Appliance Design. San Francisco: No Starch Press, 2007.
- Sobell, Mark. A Practical Guide to Linux Commands, Editors, and Shell Programming. Upper Saddle River: Prentice Hall Professional Technical Reference, 2005.
- Sobell, Mark. A Practical Guide to Red Hat. Upper Saddle River: Prentice Hall PTR, 2006.
- Stallings, W.: Data and Computer Communications, Seventh Edition, Upper Saddle River, NJ: Prentice Hall, 2003
- Stallings, William: Operating Systems. Upper Saddle River: Pearson/Prentice Hall, 2005.
- Stanfield, V.: Linux System Administration, Second Edition, Alameda, CA: Sybex Books, 2002
- Stevens, W.R.: The Protocols (TCP/IP Illustrated, Volume 1), Boston: Addison-Wesley, 1994
- Stevens, W.R.: UNIX Network Programming, Upper Saddle River, NJ: Prentice Hall, 1998
- Stutz, Michael. The Linux Cookbook, Second Edition. San Francisco: No Starch Press, 2004.
- Subramanian, M.: Network Management: Principles and Practice, Boston: Addison Wesley 2000
- Tanenbaum, A.S.: Computer Networks, Fourth Edition, Upper Saddle River, NJ: Prentice Hall, 2003
- Tanenbaum, Andrew and Albert S Woodhull: Operating Systems. Upper Saddle River: Pearson Prentice Hall, 2006
- Tanenbaum, Andrew: Modern Operating Systems. Englewood Cliffs: Prentice Hall, 2001.
- Thomas, Keir. Beginning Ubuntu Linux: from Novice to Professional, Second Edition. Berkeley: APress, 2007.
- Turnbull, James. Hardening Linux. Berkeley: APress, 2005.
- Ward, Brian. How Linux Works. City: No Starch, 2004.
- Yaghmour, Karim. Building Embedded Linux Systems. Sebastopol: O'Reilly, 2003.

ISBN 978-973-718-788-8

Material cu drept de autor